SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Marko Smolec

Zagreb, 2015.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
KATEDRA ZA STROJARSKU AUTOMATIKU

BUŠENJE NEHOMOGENIH MATERIJALA POMOĆU ROBOTA

Diplomski rad

Marko Smolec

Mentor:
Prof. dr. sc. Bojan Jerbić

Mehatronika i Robotika
Zagreb, 2015.

Izjavljujem da sam ovaj rad izradio samostalno koristeći stečena znanja tijekom studija i navedenu literaturu.

Zahvaljujem se svojem mentoru prof. dr. sc. Bojanu Jerbiću i asistentu mag. ing. Filipu Šuligoju na pruženoj stručnoj pomoći, ustupljenoj literaturi i korisnim savjetima tijekom izrade rada koji je uvelike proširio moju paletu znanja i tako me pripremio za nadolazeće izazove.

Također, zahvaljujem se svojoj obitelji i djevojci koji su mi bili velika podrška prilikom pisanja, te puni razumijevanja za moje postupke.

Marko Smolec

"Flying is learning how to throw yourself at the ground and miss." - Douglas Adams

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za diplomske ispite studija strojarstva za smjerove:
proizvodno inženjerstvo, računalno inženjerstvo, industrijsko inženjerstvo i menadžment, inženjerstvo materijala i mehatronika i robotika

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum 26 -11- 2015	Prilog
Klasa: 602-04/15-6/3	
Ur.broj: 15-1703-15-440	

DIPLOMSKI ZADATAK

Student: **MARKO SMOLEC** Mat. br.: 0035178940

Naslov rada na hrvatskom jeziku: **BUŠENJE NEHOMOGENIH MATERIJALA POMOĆU ROBOTA**

Naslov rada na engleskom jeziku: **ROBOTIC DRILLING OF INHOMOGENEOUS MATERIALS**

Opis zadatka:

Za potrebe preciznog bušenja nehomogenih materijala pomoću robota potrebno je napraviti model i simulaciju procesa bušenja ljudske kosti s višestrukim ulaznim parametrima te njegovu provjeru kroz eksperiment proveden na životinjskim kostima. Postava tehničkog sustava mora uključivati projektiranje i postavu sustava za bušenje i stege za eksperiment u radnoj okolini robota, uspostavu komunikacije između elemenata sustava i kalibraciju alata robota.

Za eksperiment bušenja homogenih struktura treba omogućiti:
- primjenu izrađenog modela za regulaciju upravljivih izlaznih parametara u realnim uvjetima,
- određivanje kritičnih parametara i njihovih optimalnih vrijednosti s obzirom na kvalitetu bušenja i kontroliranog proboja izlazne kosti.

Zadatak zadan: 24. rujna 2015.

Rok predaje rada: 26. studenog 2015.

Predvideni datum obrane: 2., 3. i 4. prosinca 2015.

Zadatak zadao: Prof. dr. sc. Bojan Jerbić

Predsjednik Povjerenstva: Prof. dr. sc. Franjo Cajner

SADRŽAJ

SADRŽAJ I
POPIS SLIKA II
POPIS TABLICA IV
POPIS OZNAKA V
SAŽETAK VI
SUMMARY VII
1. UVOD 1
2. NEHOMOGENI MATERIJALI 2
2.1. Ljudska anatomija 2
2.2. Cranium 3
2.2.1. Rasčlamba lubanje 3
2.2.2. Građa koštanog sustava 4
2.2.3. Građa kostiju lubanje 5
2.3. Bušenje Lubanje 7
2.4. Operativni zahvat bušenja lubanje 8
3. POSTAVA TEHNIČKOG SUSTAVA 10
3.1. Elemenati sustava 10
3.2. Konstrukcija 11
3.3. UR5 industrijski robot 12
3.4. IPC Integrirana električna konzola 14
3.5. Kirurška bušilica 15
3.6. Svrdla u neurokirurgiji 16
3.7. Robotiq FT 150 senzor sile 17
4. KOMUNIKACIJA 20
4.1. TCP / IP model 20
4.2. Komunikacija između računala i kontrolera robota 23
4.3. Komunikacija između računala i senzora 24
5. KALIBRACIJA 25
5.1. Kalibracija vrha alata 28
5.2. Kalibracija orijentacije alata 30
6. SIMULACIJSKI MODEL BUŠENJA KOSTI 31
6.1. Izrada simulacijskog modela 33
7. IMPLEMENTACIJA RAZVIJENOG MODELA VOĐENJA 38
8. ZAKLJUČAK 53
LITERATURA 54
PRILOZI 55

POPIS SLIKA

Slika 1. Ljudska lubanja ... 3
Slika 2. Kosti lubanje ... 3
Slika 3. Plosnate kosti ... 5
Slika 4. Presjek plosnate kosti ... 5
Slika 5. Presjek lubanje ... 6
Slika 6. Mayfieldov držač za glavu ... 8
Slika 7. Uklanjanje skalpa ... 9
Slika 8. Bušenje lubanje ... 9
Slika 9. Postava tehničkog sustava ... 10
Slika 10. Profili spojeni kutnicima i poprečna greda za veću krutost konstrukcije ... 11
Slika 11. Renderirani CAD model postolja ... 11
Slika 12. UR5 industrijski robot ... 12
Slika 13. Privjesak za učenje ... 13
Slika 14. Centralna jedinica UR5 robotske ruke ... 13
Slika 15. IPC sustav ... 14
Slika 16. Nožna kontrolna jedinica ... 14
Slika 17. Midas Rex® Legend EHS® Motor ... 15
Slika 18. Jacobs® Chuck reduktor ... 15
Slika 19. Primjer svrdla u neurokirurgiji ... 16
Slika 20. Robotiq FT 150 senzor sile ... 17
Slika 21. Ožičenje 'RS485 to RS232' pretvarača ... 18
Slika 22. Spoj senzora na kontroler robota ... 18
Slika 23. Spoj senzora na UR5 robotsku ruku ... 19
Slika 24. Usporedba OSI i TCP/IP referentnih modela ... 20
Slika 25. Protokol trostukog rukovanja ... 22
Slika 26. Poruka koju šalje kontroler robota ... 23
Slika 27. Prikaz izlaza senzora na RealTerm terminalnom programu ... 24
Slika 28. Prikaz koordinatnih sustava ... 25
Slika 29. Točka prihvata alata i koordinatni sustav središta alata ... 25
Slika 30. Gibanje alata oko definirane radne točke ... 26
Slika 31. Angle-Axis interpretacija rotacije ... 26
Slika 32. Primjer Eigen koda ... 27
Slika 33. Metoda kalibracije vrha alata ... 28
Slika 34. Ispis konzole nakon kalibracije vrha alata ... 29
Slika 35. Vrh bušilice u referentnoj točki ... 29
Slika 36. Definiranje osi gibanja alata ... 30
Slika 37. Očitanje kuteva u Polyscope korisničkom sučelju ... 30
Slika 38. Osnovna forma / blok struktura PID regulatora ... 31
Slika 39. Simulacijski model ... 31
Slika 40. Karakteristične sile i brzine vrtnje kod bušenja kosti ... 32
Slika 41. Odziv električne bušilice prilikom bušenja kosti ... 32
Slika 42. Homogena barijera / kost ... 34
Slika 43. Simulacija bušenja samo sa P regulatorom ... 35
Slika 44. Bušenje kosti sa PI regulatorom ... 36
Slika 45. Pomoćne varijable ... 36

Slika 46. Pravovremeno zaustavljanje procesa bušenja 37
Slika 47. Realniji model barijere / otpora kosti 37
Slika 48. Odziv simulacije bušenja realnijeg modela kosti 37
Slika 49. Velike oscilacije senzora sile UR5 robotke ruke 38
Slika 50. Odziv senzora sile FT 150 39
Slika 51. Filtrirani signal regulirane sile 39
Slika 52. Eksperimentalna nehomogena struktura 40
Slika 53. Kp=3 Ki=0 42
Slika 54. Kp=10 Ki=0 43
Slika 55. Kp=25 Ki=0 43
Slika 56. Kp=10 Ki=1 44
Slika 57. Kp=10 Ki=5 44
Slika 58. Kp=10 Ki=7 45
Slika 59. Bušenje životinjske kosti 46
Slika 60. Kp=10 Ki=1 46
Slika 61. Kp=12 Ki=1.5 47
Slika 62. Kp=20 Ki=3 47
Slika 63. Kp=15 Ki=2 Kd=0.01 48
Slika 64. Kp=10 Ki=0.1 Kd=0 48
Slika 65. Kp=35 Ki=1 Kd=0.01 49
Slika 66. Kp=20 Ki=0 Kd=3 49
Slika 67. Kp=2 Ki=0.01 Kd=0 50
Slika 68. Kp=12 Ki=0. 1 Kd=1 50
Slika 69. Promjena sile tokom bušenja plosnate kosti 51
Slika 70. Dijagram toka za detekciju proboja plosnate kosti 52

POPIS TABLICA

Tablica 1. Izlazne brzine reduktora 16
Tablica 2. Usporedba mehaničkih svojstava materijala.....40
Tablica 3. Utjecaji pojedinih parametara PID regulatora.....41

POPIS OZNAKA

Oznaka	Jedinica	Opis
F	N	sila
U	V	elek. potencijal, napon
m	duljina	metar
n	o/min	kutna brzina
t	s	sekunda
f	Hz	herc
v	m/s	brzina
a	m/s2	ubrzanje
m	kg	masa
k		faktor opadanja
Kp		proporcionalno pojačanje
Ki		integralno pojačanje
Kd		derivacijsko pojačanje
e		regulacijska pogreška
σ	N/mm^2	čvrstoća

SAŽETAK

Implementiranje industrijskog robota u proces kirurškog bušenja. Specijalna namjena u svrhu postizanja veće točnosti i reduciranja vremena za izvođenje samog postupka operacije, uz smanjenje troškova ulaganja zbog industrijske, a ne medicinske prirode robota. Za potrebe preciznog bušenja nehomogenih materijala napravljen je simulacijski model koji je zatim testiran na eksprerimentnom modelu, zatim na životinjskim kostima koja imaju veliku sličnost sa realnom ljudskom kosti lubanjskog svoda. Kako bi proveli postupak bušenja potrebno je dizajnirati nosivu konstrukciju, postaviti tehnički sustav, te provesti real-time komunikaciju između svih elemenata sustava. U konačnici, samo bušenje treba biti usmjereno i vođeno putem programiranje C++ aplikacije na računalu koja zahtjeva kalibraciju osi gibanja alata i same radne točke alata.

Ključne riječi: industrijski robot, TCP, regulacija, kalibracija, bušenje, nehomogeno

SUMMARY

The implementation of industrial robot in the neurosurgical procedure, drilling a hole in the skull of a living person to cure illness. We implement robot arm in order to achieve greater accuracy and reduce the time needed to perform the operation, while reducing investment costs. For the purpose of precise drilling inhomogeneous materials we created a simulation model which is tested on experimental model, then on the animal bones that have a strong resemblance to a human flat bone covering his skull. In order to carry out the process of drilling it's necessary to design the supporting structure, set up a drilling environment and conduct real-time communication between all the elements of the system. Ultimately, drilling procedure will be guided by C++ application on a PC, which also requires tool axis and TCP calibration.

Key words: industrial robot, TCP, control, calibration, drilling, inhomogeneous

1. UVOD

Medicinska pomoć za sve. Kao temelj civiliziranog društva, osnovno pravo svakog čovjeka je pravo na liječničku skrb. Povećanje uspješnosti, smanjenje trajanja postupka i veća sigurnost prilikom operativnih postupaka jest osnovni zadatak svakog liječnika, ali i inženjera strojarstva. Zašto inženjera strojarstva? Razvitkom tehnologije, međusobna povezanost strojarstva i medicine kao znanstvenih discliplina neprestano raste, a primjena robotike u operacijskim salama postaje sve veća, dok s vremenom, nezaobilazna. Idealan primjer te tvrdnje je robotsko bušenje u neurokirurgiji, kao veliki potencijal razvitka robotike upravo u smjeru od kojeg će medicina, ali i čovječanstvo imati velike koristi. Ovaj diplomski rad je dokaz toga.

Ako uzmemo u obzir bitnu činjenicu da se prilikom operativnog zahvata ne može izbjeći oštećenje tkiva, glavni zadatak operativnog postupka postaje minimalizacija oštećenja, što je moguće veća pošteda živčanih struktura, što će za svoju posljedicu imati manji broj komplikacija te naravno, brži oporavak. Otkako je dokazano da ne postoje nijeme zone mozga, da svaki dio, svaka regija i zona imaju svrhu i smisao, taj zadatak je postao prioritetan prilikom svakog planiranja operacije u neurokirurgiji. U konačnici, treba imati na umu da se ne operira neki dio tkiva, neka anomalija ili cista, operira se ljudski um. Upravo zato 'key-hole' pristupi i endoskopske tehnike svakodnevno osvajaju sve veći dio 'tržišta'.

Roboti se odlikuju nizom prednosti nad čovjekom, čime je neosporiva njihova važnost u operacijskoj sali. Bolja orijentacija (pogotovo kada je robot integriran sa suvremenim radiološkim tehnikama) u trodimenzionalnom prostoru i mnogo veća preciznost prilikom izvršavanja zadataka samo su neke od prednosti, ali kao glavna, ističe se činjenica da su njegove performanse i način izvršavanja zadaće uvijek isti, dok kod čovjeka ovise o brojnim čimbenicima, poput umora i stresa, koji imaju velik utjecaj na njegov rad i koncentraciju.

Bušenje lubanje zahtjevan je proces koji zahtijeva veliku preciznost, jer samo malo odstupanje može rezultirati velikim i nepotrebnim oštećenjem tkiva. Isto tako, pravovremeno zaustavljanje, bez da se ošteti moždana ovojnica od ključne je važnosti. Kada je slučaj bušenja lubanje, konstantan pritisak bušilice, bez obzira na promjenu njegove nehomogene strukture, je nemoguć pothvat i za najvještijeg kirurga. Ali ne i za robota. Ovisno o starosti, povijesti bolesti i drugim uvjetima, čvrstoća i struktura kosti se jako razlikuje. Također, makar se buši kost koja je u prosjeku debljine oko 6 milimetara, vrijeme bušenja ne smije biti dugotrajno jer riskiramo veliki porast temperature, a povišena temperatura, npr. od 42°C uzrokuje poremećaj metabolizma kosti, a temperatura od 47 °C tijekom 1 minute izaziva ireverzibilne promjene kako građe, tako i funkcije kosti.

Dok Znatiželja* buši na Marsu, odvažnost buši na Zemlji. Projekt RONNA.

* Robotsko vozilo 'Curiosity' američke NASA-e buši stijene na površini Marsa

2. NEHOMOGENI MATERIJALI

Nehomogeni materijali su materijali čija je gustoća (specifična masa) promjenjiva i zavisi od položaja pojedinih čestica mase promatranog tijela. Sam proces bušenja robotskom rukom je složen jer zahtjeva preciznu kontrolu i vođenje, a nehomogena struktura predmeta bušenja čini proces još složenijim. Kao idealan proces koji zahtjeva složeno bušenje, jest bušenje u neurokirurgiji, a kao idealan predmet bušenja uzimamo ljudsku anatomiju, odnosno ljudsku kost, koja je po svojoj strukturi tipičan primjer nehomogenosti materijala kojim se ovaj rad bavi.

2.1. Ljudska anatomija

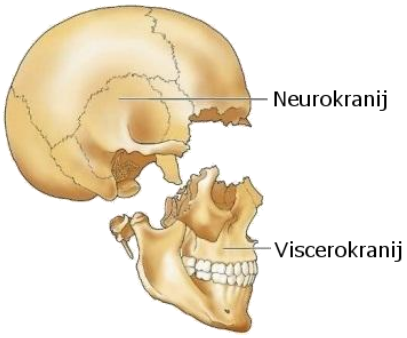
Kako bi se moglo u potpunosti razumjeti sam postupak bušenja u medicini, potrebno je razumjeti osnovnu medicinsku terminologiju i dio anatomije čovjeka koja je uključena u ovaj rad. Tijelo čovjeka je podijeljeno na tri osnovne cjeline, na trup (lat. *truncus*), te na gornje i donje udove (lat. *membra superiora et inferiora*). Trup se dalje raščlanjuje na glavu (lat. *caput*), vrat (lat. *collum*) i trup u užem smislu. Pošto se rad specifično bavi glavom, tj. bušenjem lubanje, biti će dani opisi i bitne činjenice vezane isključivo za nju.

Osnovne osi tijela su uzdužna (*vertikalna /* eng. *longitudinal*), poprječna (*transverzalna /* eng. *frontal*) i sagitalna os (lat. *sagitta = strijela /* eng. *sagittal*). Pri uspravnom stavu uzdužna os je okomita na podlogu, poprječna os tijela je okomita na uzdužnu, dok je sagitalna os usmjerena od stražnje prema prednjoj strani tijela i ujedno je okomita na uzdužnu i poprječnu os. Najvažniji termini u medicini koji se tiču usmjerenja u prostoru, te su vezana za područje koje obrađuje ovaj rad su:

- kranijalno, lat. *cranialis, -e* = u smjeru lubanje
- superior, *-ius* = prema gore pri uspravnom stavu tijela
- inferior, *-ius* = prema dolje pri uspravnom stavu tijela
- medijalno, lat. *medialis, -e* = prema sredini, mediosagitalnoj ravnini
- lateralno, lat. *lateralis, -e* = od sredine (mediosagitalne ravnine) prema van
- medius, *-a, -um* = u sredini
- *medianus* = u mediosagitalno ravnini
- dubinski, *profundus, -a, -um* = prema unutrašnjosti tijela

2.2. Cranium

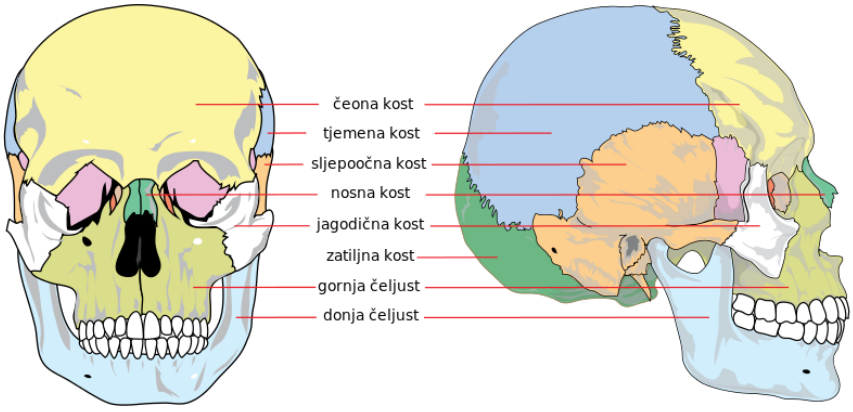
Predmet bušenja ovog diplomskog rada jest lubanja, što je pravi izazov zbog njene nehomogene strukture. Ljudska lubanja (lat. *cranium*) je koštana osnova glave, oslonjena na vrh kralježnice, ovalnog je oblika i šira straga nego sprijeda. Uz to što tvori osnovu lica, lubanja služi kao oklop za mozak i osjetne organe. Dijeli se na dva dijela, dio u kojemu je smješten mozak (neurocranium) i dio koji čine kosti lica, takozvani visceralni dio (viscerocranium). Granica između dva dijela lubanje proteže se od područja korijena nosa, gornjim rubom orbite, te doseže do vanjskoga slušnog hodnika. Dakle, viscerokranij se nalazi ispred i ispod neurokranija. Lubanja je sastavljena je od niza plosantih i nepravilnih kostiju koje su sve, izuzev donje čeljusti, nepomično uzglobljene. [11]



Slika 1. Ljudska lubanja

2.2.1. Rasčlamba lubanje

Kao što je već navedeno, lubanja se sastoji od dvaju dijelova, neurokranija i viscerokarnija. Razlikujemo svod lubanje (lat. *calvaria*) i razmjerno ravnu bazu lubanje (lat. *basis cranii*). Također sadrži i potpornje koji raspoređuju živčani tlak, a početni je dio za prolazak zraka i hrane. Oblik lubanje ovisi dijelom o mišićima, koji mogu svojim djelovanjem uvjetovati određene promjene, te drugim dijelom o sadržaju struktura unutra lubanje, te tako postoji međuodnos između kostiju neurokranija i mozga unutra njega.



Slika 2. Kosti lubanje

Ljudska lubanja sastoji se od 22 kosti, od kojih je u odraslih osoba 21 čvrsto spojena šavovima tako da se teško mogu izolirati. Jedino je donja čeljust povezana zglobno sa sljepoočnom kosti. Lubanjske kosti su većinom pločaste kosti, a sudjeluju u omeđenju šupljina u kojima su smješteni mozak i osjetni organi.

Kosti ljudske lubanje:

- neurokranij (lat. *neurocranium*) u kojemu je smješten i zaštićen mozak, čini 8 kostiju:
    - zatiljna kost
    - 2 tjemene kosti
    - čeona kost
    - 2 sljepoočne kosti
    - klinasta kost
    - rešetnica
- kostur lica (lat. *splanchnocranium* ili *viscerocranium*), čini 14 kostiju:
    - 2 nosne kosti
    - 2 gornje čeljusti
    - 2 suzne kosti
    - 2 jagodične kosti
    - 2 nepčane kosti
    - 2 donje nosne školjke
    - raonik
    - donja čeljust

Donja nosna školjka, suzne kosti, nosne kosti i raonik se zbog svog razvoja također mogu smatrati kostima neurokranija.

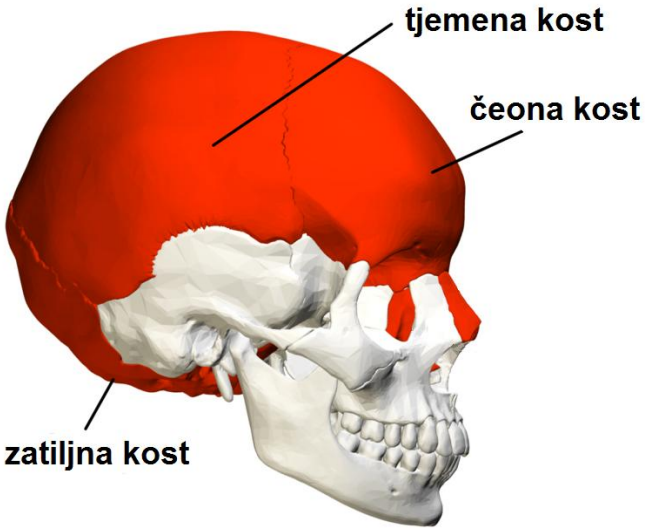
2.2.2. Građa koštanog sustava

Kosti lubanje zbog svoje građe ne sudjeluju u produkciji krvi. Koštana krv, koja se nalazi u dugim ili cjevastim kostima, stvara krvne ćelije. Kost se sastoji od koštanih stanica i minerala odloženih u organskom matriksu. Koštano tkivo sadrži tri vrste stanica, osteoblaste, osteocite i osteoklaste.

Osteoblasti nastaju iz vezivne strome, uvijek su smješteni na površini koštanog tkiva, a izlučuju bjelančevine. Kada se osteoblast potpuno okruži tek izlučenim matriksom postaje osteocit (najbrojnije stanice koštanog sustava). Smješten je u lakuni i povezan citoplazmatskim izdancima sa okolnim osteocitima. Citoplazmatski izdanci su u međusobnoj komunikaciji putem tijesnih spojeva kojima hranjive tvari dolaze u stanice. Osteoklasti su stanice potrebne za resorpciju kosti.

2.2.3. Građa kostiju lubanje

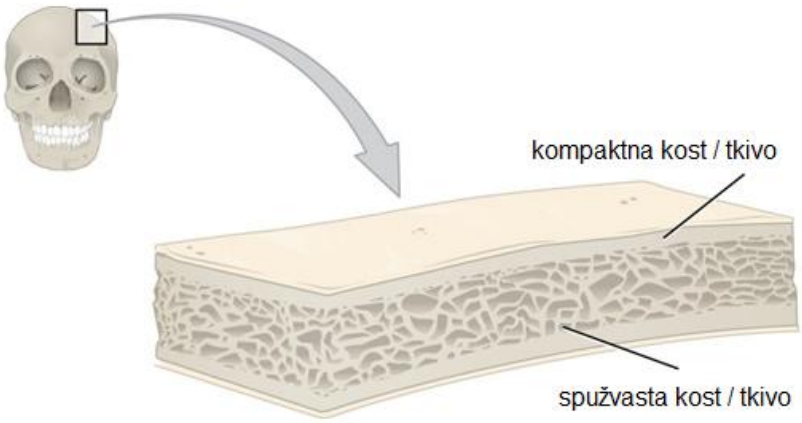
Kod bušenja lubanje najčešće se buše tjemena (dvije su tjemene kosti), zatiljna i čeona kost. Te kosti su po strukturi plosnate kosti (eng. *flat bone*). Plosnate kosti su tanke, čvrste ploče, a sastoje se od tri dijela.



Slika 3. Plosnate kosti

Plosnate kosti lubanje sastoje se od:

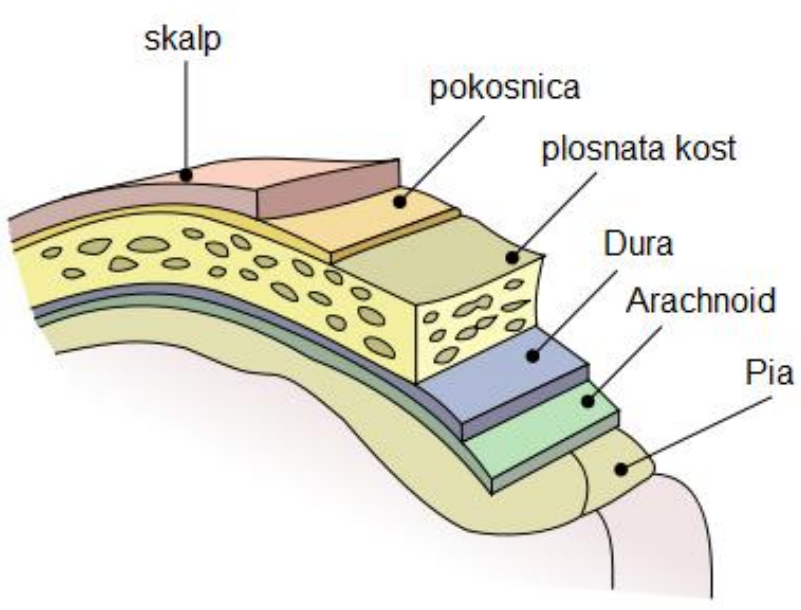
- vanjskog, kompaktnog sloja, (lat. *lamina externa*)
- unutarnjeg, kompaktnog sloja, (lat. *lamina interna*) i
- međusloja, spužvastog sloja (eng. *diploë, spongy bone*)



Slika 4. Presjek plosnate kosti

Kompaktna kost je homogena, s lamelama i sadrži kanale za krvne žile. Unutrašnjost čine spongioza i koštana srž. Spužvasta kost građena je od koštanih gredica čiji položaj prati smjer djelovanja sile na kost. Takva građa čini kost laganom, elastičnom i čvrstom strukturom koja podupire tijelo i omogućuje kretanje. Također, spužvasta kost sadrži mnogobrojne vene. Gubitak koštane mase i poremećaj mikrostrukture očituje se slabljenjem ponajprije spužvaste kosti koja postaje rahla i šupljikava.

Lubanja je izvana prekrivena skalpom kojeg krvlju snabdjeva pet pari arterija, te se upravo zato prilikom rezanja pojavljuje velika količina krvi. Ispod skalpa se nalazi pokosnica (eng. *pericranium*). Pokosnica je vezivna opna koja prekriva vanjsku površinu kostiju. Njenim uklanjanjem dolazimo do plosnate kosti koja se buši. Ispod kosti se nalaze tri zaštitne opne, Dura, Arachnoid i Pia mater. Mater dolazi od riječi majka, jer opne kao majka obavijaju mozak. Te tri moždane ovojnice (meninge) se nastavljaju na ovojnice leđne moždine. Dura mater je tvrda vanjska ovojnica koja je čvrsto srasla uz kosti lubanje. Arachnoidea je mekša od dure te paučinasta, svojim izgledom podsjeća na paukovu mrežu. Pia mater je meka ovojnica koje je usko vezana uz mozak. Između paučinaste i meke ovojnice se nalazi subarahnoidalni prostor koji je ispunjen likvorom i kroz taj prostor prolaze veće krvne žile mozga. Likvor je bistra, cerebrospinalna tekućina koja okružuje središnji živčani sustav.



Slika 5. Presjek lubanje

2.3. Bušenje Lubanje

Općenito, bušenje je obrada odvajanjem čestica kod koje alat vrši glavno gibanje, rotaciju, i posmično gibanje, translaciju, a služi za izradu rupa i provrta. Kao takvo, bušenje u medicini i strojarstvu je isto, ali uz puno veću složenosti i dozu opasnosti. Otvaranje lubanje je jedna od najstarijih poznatih operacija, obavljana od prahistorijskog doba, a vršena je trepanom ili trefinom u magijske, terapeutske ili neke druge svrhe, kojom se uklanjao okrugli ili četvrtasti segment sa kosti svoda lobanje. Takav postupak se naziva trepanacija (lat. trepanatio, svrdlo).

Kraniotomija (eng. *craniotomy*) je termin kojim označujemo neurokirurški zahvat kojim izrezujemo dio koštanog svoda lubanje, kako bismo pristupili strukturama unutar lubanje. Kraniotomije se mogu sastojati od samo jednoga trepanskog otvora (eng. *burr hole*) ili se podiže koštani poklopac različitih dimenzija, kada je potreban širi pristup.

Burr hole upotrebljavamo za:

- ugradnju kranijalnog dijela shunta kod hidrocefalusa
- ugradnju elektroda za stimuliranje dubokih moždanih jezgara
- pri uvođenju katetera za mjerenje intrakranijskoga tlaka
- stereotaktičke biopsije tumora
- stereotaktičke aspiracije hematoma, apscesa, intracerebralnih cisti
- uvođenje endoskopa intrakranijalno.

2.4. Operativni zahvat bušenja lubanje

Prije samog operativnog zahvata bušenja lubanje potrebno je obrijati kosu s vlasišta (ujutro neposredno prije operacije), nakon čega slijedi energično mehaničko pranje kože skalpa. Sam postupak bušenja lubanje dijeli se na 3 faze, od kojih svaka zahtjeva veliku dozu koncentracije.

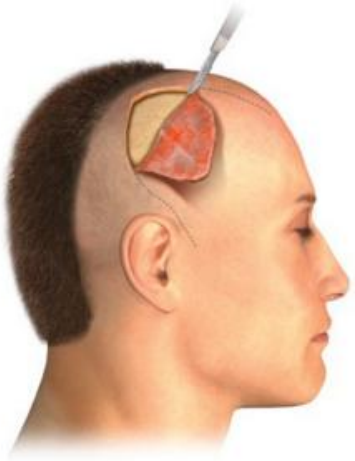
Faza I

Pacijent je uspavan, postavljen na operacijski stol, glava je učvršćena u posebno konstruirani držač za glavu (eng. *Mayfield skull clamp*). Držač ima oblik trozube hvataljke ili kliješta s oštrim vrhovima, koji probijaju tabulu eksternu kosti i toliko čvrsto obuhvaćaju lubanju da su i najmanji pokreti glave tijekom operacije nemogući. Nužna je potpuna nepokretljivost glave, naročito kod bušenja lubanje, kada i najmanja rotacija ili mijenjanje kuta može odvesti kirurga daleko od cilja. Na koži skalpa olovkom se označava crta po kojoj će se napraviti rez skalpelom. Rez skalpelom se radi u smjeru mišića, zbg čega rana brže zacjeljuje. Također, od velike je važnosti odvojiti kožu od mjesta bušenja jer je koža jako osjetljiva na termička oštećenja.

Slika 6. Mayfieldov držač za glavu

Faza II

Druga faza počinje nakon što je uklonjen skalp. Prije toga je potkožno tkivo infiltrirano fiziološkom otopinom s 1:1000 otopinom adrenalina da bi se smanjilo krvarenje koje može biti veoma obilno iz dobro prokrvljene kože poglavine. Pažljivo se kauteriziraju sve krvne žile – ako se to ne učini u ovoj fazi, poslije vazospazam popušta i pacijent može, krvareći cijelo vrijeme operacije iz poglavine, izgubiti znatne količine krvi. Kauterizacija se koristi za zaustavljanje krvarenja mali krvih žila. U neurokirurgiji se koristi bipolarni mod kauterizacije. Poglavina i mišići pažljivo se odvoje od kosti.



Slika 7. Uklanjanje skalpa

Faza III

Posebnom bušilicom, koja se automatski zaustavlja kada je kost probijena, načini se otvor. Komadići kosti ostaju priljepljeni na vrhu svrdla, pa samim uklanjanjem bušilice, uklanjamo i njih. Daljnji postupak ovisi o kirurškoj operaciji koja se provodi.

Slika 8. Bušenje lubanje

3. POSTAVA TEHNIČKOG SUSTAVA

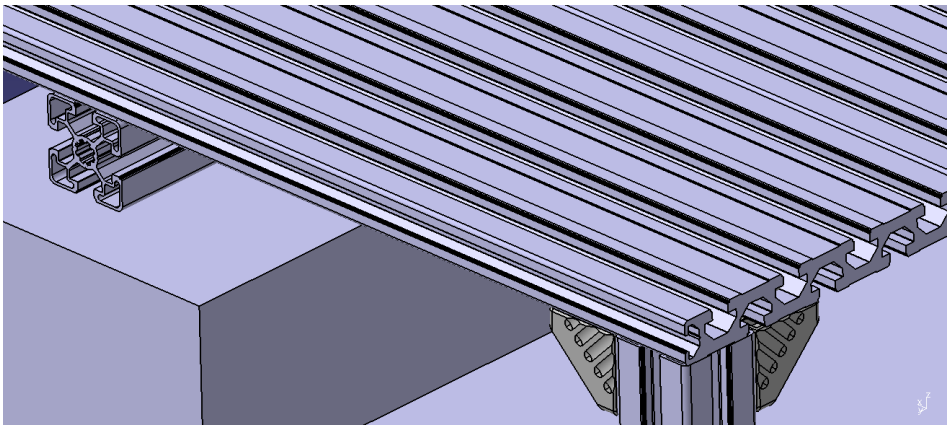
3.1. Elemenati sustava

Sustav za bušenje nehomogenih materijala se sastoji od UR5 industrijskog robota, IPC integrirane električne konzole (sa svim pripadajućim elementima), senzora sile i same mobilne konstrukcije na kojoj je sve postavljeno. Opis svakog od elemenata dan je u narednim poglavljima.

Slika 9. Postava tehničkog sustava

3.2. Konstrukcija

Za potrebu ovog diplomskog rada, dizajnirana je posebna kontrukcija koja, uz sve prednosti klasične izvedbe, nudi i dozu mobilnosti, što je potrebno u ograničenim prostorima laboratorija, ali i kod bušenja modela koji zahtijevaju posebnu orijentaciju robotske ruke. Postolje je konstruirano u programskom paketu Catia uz fleksibilne parametre, prilagodljive prema elementima koje će konstrukcija nositi. Također, postolje ima mogućnost da se učvrsti na zid kako bi se postigla veća stabilnost. U podnožju konstrukcije nalazi se protuuteg od 100kg. Protuuteg služi da bi se neutralizirao utjecaj razvoja velikog momenta inercije uslijed naglog usporavanja / ubrzavanja. Radni prostor postolja, gdje je montiran robot, veličine je $1m^2$. Iznad protuutega nalazi se prostor predviđen za kontroler robota i kučište računala kojim se upravlja robotom.



Slika 10. Profili spojeni kutnicima i poprečna greda za veću krutost konstrukcije

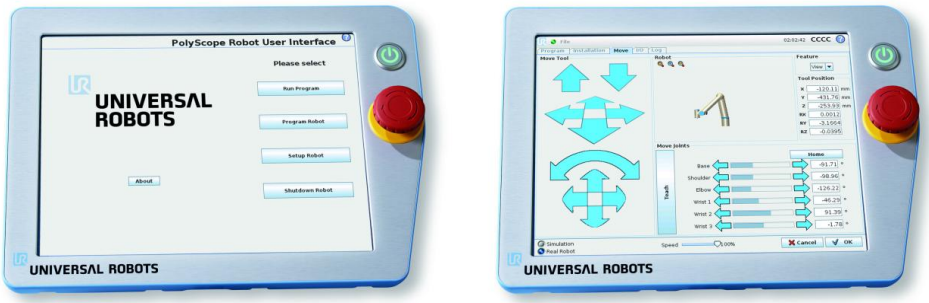
Slika 11. Renderirani CAD model postolja

3.3. UR5 industrijski robot

Slika 12. UR5 industrijski robot

Robotska ruka korištena u diplomskom radu jest UR5 robotska ruka poznatog Danskog proizvođača Universal Robots. Universal Robots je kompanija koja zauzima sve veći dio tržišta kao konkurencija velikim svjetskim proizvođacima industrijskih robota poput Kuke i FANUC-a. Također, surađuje sa kompanijama poput as Lear Corporation, Franke , Oticon, Johnson, Clamcleats, Kunshan Dongwei, VW i BMW.

UR5 industrijski robot je robotska ruka sa šest stupnjeva slobode gibanja opće namjene, a zbog jednostavnosti i 'user friendly' sučelja moguće ju je primjeniti u bilo kojem zadatku automatizacije bez poteškoća. Kao software koristi Linux OS platformu sa posebnim programskim sučeljem 'PolyScope' koji je jako sličan C++ programskom jeziku, ali je jednostavniji, što je velika prednost po pitanju složenosti instaliranja UR5 industrijskog robota u postrojenje, tj. u proizvodni proces, uz koji ima privjesak za učenje (eng. teach pendant) osjetljiv na dodir. Također, radni prostor robota, radijusa 850mm i ponovljiva gibanja sa teretom do 5kg, samo su još neke od prednosti ovog robota.



Slika 13. Privjesak za učenje

Na centralnoj jedinici UR5 robotske ruke, kontroleru, istovremeno je aktivno više servera, svaki prilagođen određenom tipu komunikacije i rada. Server pod portom 30003 nudi 'Real Time Client Interface' model rada, što u prijevodu znači da je moguće frekvencijom od 125Hz aktivno dobivati sve informacije o stanju robota, njegovom položaju, silama u zglobovima, struji i sl. Navedene informacije su od ključne važnosti za aktivno i točno vođenje robota, pa i same operacije bušenja zbog određivanja pravilnih parametara. Kinematički model robota je izveden u Denavit-Hartenberg (DH) notaciji. Transformacija između različitih koordinatnih sustava robota ostvaruje se množenjem matrica transformacije.

Slika 14. Centralna jedinica UR5 robotske ruke

3.4. IPC Integrirana električna konzola

Slika 15. IPC sustav

IPC® sustav je sustav široke namjene koji je primjenjiv u uklanjanju mekog ili tvrdog tkiva, te kosti. Električnu konzolu je zbog njenoga oblika i veličine lako ugraditi u bilo koji sustav. Isto tako, uz veliki spektar operacija u kojima je primjenjiva ima veliki broj dodataka i specijaliziranih alata. U istom trenutku mogu biti aktivna 4 alata. IPC sustav korišten u ovom radu sastoji se od električne konzole, nožne kontrolne jedinice i alata za bušenje.

Nožna kontrolna jedinica (FCU) je višenamjenski nožni prekidač koji omogućava kontrolu odabira ručnog instrumenta, brzinu vrtnje ručnog instrumenta, način rada i smjer vrtnje. Brzinu je moguće regulirati u ovisnosti o jačini pritiska pedale ili u drugom modu rada, samim pritiskom pedale postići maksimalnu brzinu vrtnje. Također, dno pedale je posebno postavljeno da bi se izbjegla mogućnost proklizavanja u operativnoj sali.

Slika 16. Nožna kontrolna jedinica

3.5. Kirurška bušilica

Kirurška bušilica se sastoji od 2 dijela, od motora i reduktora.

Midas Rex® Legend EHS® Motor je električni motor koji može razviti veliki okretni moment, a predviđen je za seciranje kosti i biomaterijala. Radni opseg motora je od 200 do 75,000 okretaja u minuti (eng. *RPM*). Motor ima opseg od 2.03 cm i dugačak je 9.02 cm, mase svega 180 g. Da bi se izbjeglo pregrijavanje i moguće oštećenje motora, predviđen je maksimalan konstantni rad u trajanju 10 minuta pri 75,000 okretaja, nakon čega treba uslijediti mirovanje u trajanju od 25minuta (na temperaturi od 20°C). Pneumatske bušilice se puno češće koriste u praksi od električnih bušilica zbog specifičnosti uvijeta u kojima se upotrebljavaju i potrebe za sterilizacijom prije svakog zahvata, te zbog manje osjetljivosti dijelova na povišenu temperaturu. Isto tako, pneumatske bušilice se najčešće koriste u oralnoj medicini.

Slika 17. Midas Rex® Legend EHS® Motor

Jacobs® Chuck reduktor i nastavak za kirurška svrdla. Reduktor čine planetarni prijenosnici. Planetarni prijenosnici vrsta su zupčaničkih prijenosnika u kojima neki zupčanici (tzv. planeti) uz rotaciju oko vlastite osi izvode još i kružno gibanje po unutarnjem obodu središnjeg (tzv. sunčanog) zupčanika. Reduktor na izlazu daje smanjeni broj okretaja prema tablici danoj ispod.

Slika 18. Jacobs® Chuck reduktor

Tablica 1. Izlazne brzine reduktora

Brzina podešena na konzoli	AD01 izlazna brzina (max)	AD03 izlazna brzina (max)
60,000 rpm	645 rpm	830 rpm
70,000 rpm	745 rpm	965 rpm
72,000 rpm	770 rpm	995 rpm
74,000 rpm	790 rpm	1020 rpm
75,000 rpm	805 rpm	1035 rpm

3.6. Svrdla u neurokirurgiji

Svrdla u medicini imaju jednaku važnost kao i u strojarstvu. Bilo u ortopediji ili neurokirurgiju, bušenje kostiju igra veliku ulogu i zahtjeva razinu vještine i iskustva. Za razliku od nekih medicinskih pomagala i uređaja, za svrdla nije od presudne važnosti da budu otporna na koroziju, jer ne ostaju u dugom vremenskom kontaktu sa tjelesnim tekućinama. Od mnogo veće važnosti je da se rezna oštrica svrdla ne istoši ili ošteti prilikom većeg broja bušenja, tj. da što duže zadržava svoja početna svojstva, pogotovo nakon višekratnih sterilizacija u autoklavu (autoklav je grijana hermetička naprava za sterilizaciju medicinskog pribora).

U medicini su propisani posebni standardi za svrdla, DIN 1.4112 ili AISI 440B. Najčešće se sastoje od 0.85% ugljika, 18% kroma, 1% molibdena, 1%mangana, 1 % silicija i 78% čelika. Kao što je rečeno, za svrdla je najbitnije da što duže zadrže svoja probitna svojstva, tj. da imaju što duži vijek trajanja. Temperatura je jedan od najbitnijih parametara koji utječu na vijek trajanja svrdla. Kod bušenja plosnate kosti temperatura ne igra veliku ulogu, iz više razloga. Mala debljina kosti, u prosjeku 5.6mm, te kratko vrijeme bušenja ne dopuštaju da se razvoj visokih temperatura koje bi dovele do termičke nekroze kosti (*termička osteonekroza*), oštećenja kosti uslijed povišene temperature. Isto tako, tokom bušenja vrši se hlađenje mjesta bušenja fiziološkom otopinom. U medicini, pogotovo kod bušenja puno debljih kostiju, pravilno podmazivanje je od iste važnosti kao i kod bilo koje strojne obrade.

Slika 19. Primjer svrdla u neurokirurgiji

3.7. Robotiq FT 150 senzor sile

Slika 20. Robotiq FT 150 senzor sile

Robotiq FT 150 senzor je šest osni senzor sile i momenta kojega je vrlo jednostavno montirati i ističe se velikom otpornošću na šum. Nema potrebe za pretvorbom izlaznog signala senzora, koristi jednostavnu plug-and-play tehnologiju koja je kompatibilna sa Universal robotima. Također, za senzor postoji besplatni programski paket za Linux i Windows koji uvelike pomažu prilikom razvoja aplikacija.

Mjerni opseg senzora je od ± 150N za silu u smjeru svake osi, te ± 15Nm za moment uz šum od max 0.5N, tj. 0.03Nm. Senzor šalje podatke frekvecijom od 100Hz (6000 puta u minuti) što je dovoljno da bi se postiglo pravovremeno zaustavljanje prilikom bušenja uz konstantni posmak. Robot šalje podatke računalu putem TCP veze brzinom od 125Hz. Sam senzor je težak 650g, a njegova optimalna temperatura rada je od 15°C do 35°C.

Senzor se spaja na M12, 5 pinski (IEC 61076-2-101) kabel koji se spaja na 'RS485 to RS232' pretvarač. Od velike je važnosti da se pinovi ne spoje krivo jer se tako može oštetiti senzor, a spaja se prema shemi na Slici 21. 24V i GND. Žice se spajaju direktno na istoimene pinove kontrolera robota, dok se 'RS232 to USB' spaja u USB port na istom kontroleru. U konačnici, uključuje seo UR5 robot, te se vrši inicijalizacija i pomoću USB-a instaliraju se driveri senzora. Uz driver se instaliraju i testni programi kojima se može provjeriti ispravnost senzora. Također, senzor se može direktno preko 'RS232 to USB' kabela spojiti na računalo i čitati podatke.

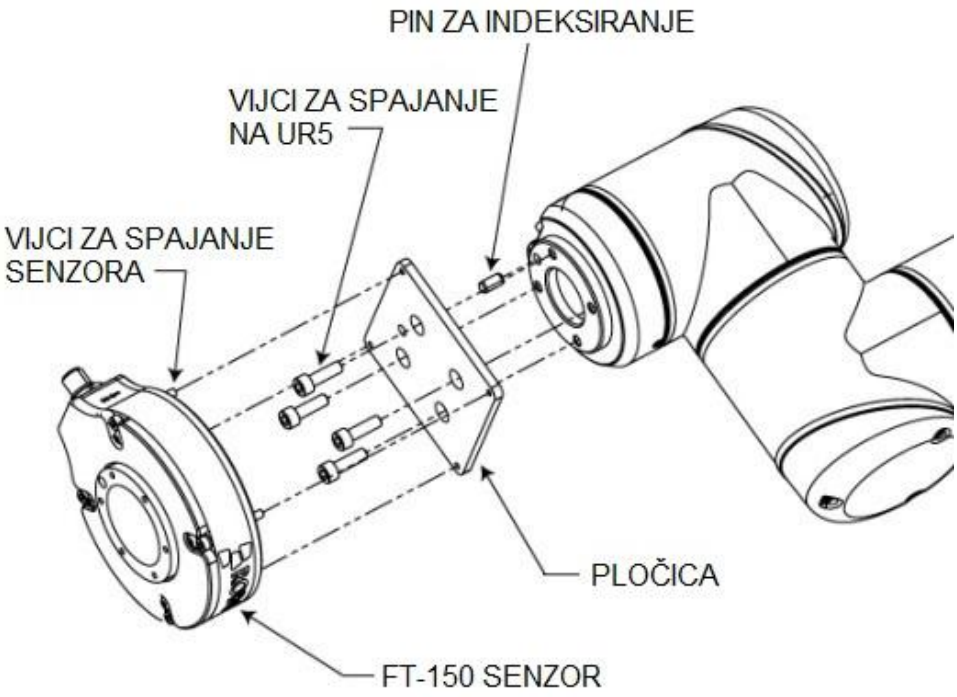
RS 232 SIDE	
PIN	SIGNAL
1	-
2	OUTPUT (TX)
3	INPUT (RX)
4	-
5	GND
6	-
7	-
8	-
9	-

RS 485 SIDE	
PIN	SIGNAL
1	485+
2	485-
3	DO NOT USE (5V IN)
4	GND

Slika 21. Ožičenje 'RS485 to RS232' pretvarača

Slika 22. Spoj senzora na kontroler robota

Druga pločica, ona koja povezuje prihvatnicu kirurške bušilice i senzor, morala je biti konstruirana te je dana na izradu. Nacrt pločice je dan u prilogu.



Slika 23. Spoj senzora na UR5 robotsku ruku

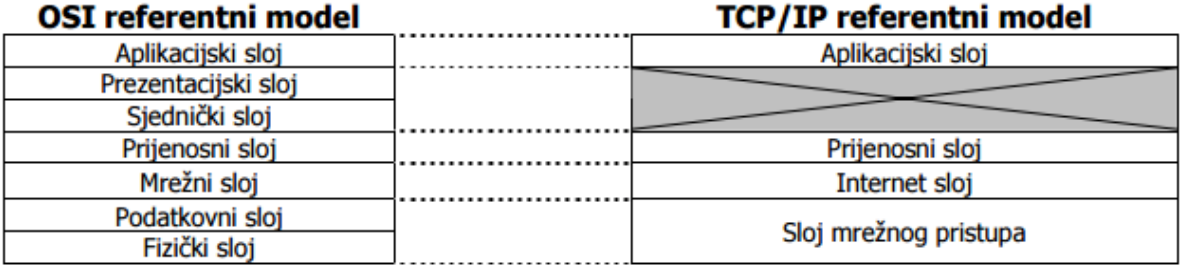
4. KOMUNIKACIJA

4.1. TCP / IP model

Istraživački odjel ministarstva obrane SAD (DARPA - Defense Advanced Research Projects Agency) 1969. godine pokrenuo je projekt uspostave eksperimentalne mreže s prespajanjem paketa nazvane ARPANET, čija je namjena bila ispitati tehničke mogućnosti razmjene podataka pomoću računalne mreže. Već 1975. godine ARPANET je od eksperimentalne postala operativna mreža. Tada se javlja TCP/IP skup protokola koji je usvojen kao vojni standard 1983. Korištenje tog standarda bio je preduvjet za spajanje na ARPANET. DARPA je potakla ugradnju TCP/IP u operacijski sustav UNIX i time je stvorena prva veza između operacijskog sustava UNIX (Sveučilište Berkley-BSD UNIX) i TCP/IP skupa protokola. U vrijeme kad je TCP/IP postao standard počeo se pojavljivati termin Internet, a mreža ARPANET podijeljena je na dva dijela: MILNET (dio podatkovne mreže ministarstva obrane SAD) i manji ARPANET. TCP/IP skup protokola prihvaćen je kao standard zbog pogodnosti koje je jedini u danom trenutku nudio, neki od njih su:

- Neovisnost o tipu računalne opreme i operacijskih sustava, te o pojedinom proizvođaču
- Neovisnost o tipu mrežne opreme na fizičkoj razini i prijenosnog medija, što omogućava integraciju različitih tipova mreža (Ethernet, Token Ring, X.25...)
- Jedinstveni način adresiranja koji omogućava povezivanje i komunikaciju svih uređaja koji podržavaju TCP/IP
- Standardizirani protokoli viših razina komunikacijskog modela, što omogućava široku primjenu mrežnih usluga [3]

Većina mreža organizirana je kao niz slojeva ili nivoa (eng. *layers, levels*), te je svaki sloj izgrađen nad onim ispod njega bez kojeg nije funkcionalan. Broj nivoa, imena, sadržaji i funkcije slojeva razlikuju se kod različitih vrsta mreže, odnosno različitih modela rada mreže nastalih tijekom povijesnog razvoja od strane različitih proizvođača računalne i mrežne opreme. Nemogućnost komunikacije različitih mrežnih modela dovela je do velikih problema u usklađivanju njihove međusobne komunikacije.



Slika 24. Usporedba OSI i TCP/IP referentnih modela

Svaki sloj ima svoju strukturu podataka i terminologiju koja opisuje tu strukturu. Unatoč tome što je OSI referentni model općenito prihvaćen, povijesni i tehnički otvoreni standard Interneta je TCP/IP. TCP/IP referentni model i TCP/IP stog protokola čine mogućom podatkovnu komunikaciju između bilo koja dva računala na svijetu skoro brzinom svjetlosti. TCP/IP model ima svoju povijesnu važnost, baš kao i standardi koji su omogućili da procvjetaju telefoni, uporaba električne struje, željeznica, televizija i sl.

TCP (eng. *Transmission Control Protocol*) dizajniran je tako da osigura pouzdani tok bajtova s jednog do drugog kraja po nepouzdanim mrežama koje se mogu razlikovati po topologiji, propusnosti, kašnjenju, veličini paketa ili nečemu drugom. Svako računalo koje podržava TCP ima TCP prijenosni entitet (dio softvera) kao korisnički proces ili kao dio jezgre koja upravlja TCP tokovima (eng. *streams*). TCP entitet prihvaća od lokalnih procesa tok podataka korisnika, razbija ga u dijelove koji ne prelaze 64 kB i šalje svaki dio kao posebni IP datagram koji kad stigne na drugi kraj, predaje se TCP entitetu koji obnavlja originalni tok bajtova.

TCP usluga se osigurava tako da i pošiljatelj i primatelj kreiraju krajnje točke (eng. *end points*) tj. spojne točke (SOCKET) od kojih je svaka definirana IP adresom host-a i brojem port-a (16 bit-ni broj za host i identifikacijska oznaka usluge). Da bi se osigurala TCP usluga, mora se uspostaviti veza između socket-a na host-u koji šalje i socket-a na host-u koji prima podatke. Brojevi port-ova ispod 256 (well-known ports) rezervirani su za standardne usluge, a svaki host može za sebe odlučiti kako će dodijeliti ostale svoje port-ove. Šaljući i primajući TCP entiteti razmjenjuju podatke u obliku SEGMENTA podataka. Segment se sastoji od 20 byte-nog TCP zaglavlja (uz opcionalni dio) za kojim slijedi 0 ili više byte-ova podataka, a nastaje skupljanjem podataka od nekoliko upisivanja (eng. *write*) ili razbijanjem podataka od jednog upisivanja. Svaki segment uključujući i TCP zaglavlje mora stati u 65535 byte-ova IP paketa. Ako je segment prevelik za mrežu kroz koju mora proći, router vrši fragmentaciju u više manjih segmenata od kojih svaki dobiva svoje IP zaglavlje.

Format TCP segmenta se sastoji se od:

- zaglavlja: ima 20 byte-ni fiksni dio i opcionalni dio varijabilne dužine.
- podataka: 65494 byte-ova (65535 B - 20 B IP zaglavlja - 20 B TCP zaglavlja), segmenti sa 0 byte-ova podataka koriste se za potvrde i kontrolne poruke.

Zaglavlje sadrži desetak polja koja počinju s izvornim port-om (16 bit-a) i odredišnom port-om (16 bit-a) te slijedi redni broj TCP veze i druga.

TCP veza je tok byte-ova, a ne tok poruka, odnosno TCP softver ne zna što znače byte-ovi koje prenosi i ne čuva granice poruka već ih šalje kao segmente koji se sastoje od grupe byte-ova. Osnovni protokol kojeg koriste TCP entiteti radi na načelu da nakon slanja segmenta pošiljatelj pokreće svoj timer (vremenski brojač). Kad poslani segment stigne na odredište primalac šalje u segmentu potvrdu - broj koji odgovara broju slijedećeg segmenta kojeg očekuje primiti. Ako timer istekne prije nego što je primljena potvrda, segment se šalje ponovo. [4]

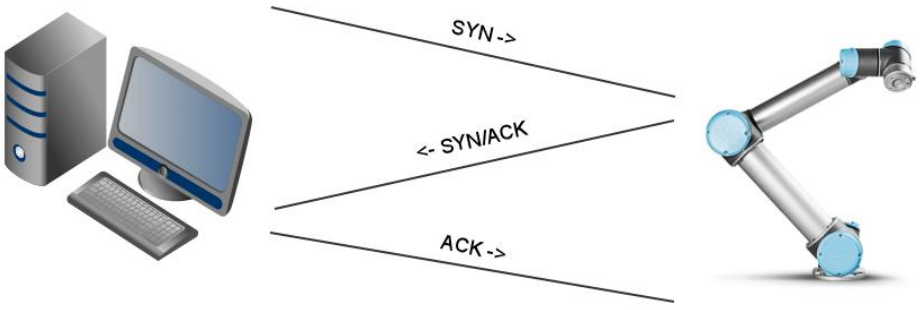
Za uspostavljanje komunikacije TCP koristi tzv. protokol trostrukog rukovanja, sinkronizacija u tri koraka (uspostava spoja, raskid spoja, *eng*. Three way Handshake). Prije nego li se klijent pokuša spojiti sa poslužiteljem, poslužitelj se mora povezati i slušati svoj ulaz (*eng*. port), te ga otvoriti za moguću vezu (*eng*. passive open). Kada je 'passive open' uspostavljen, klijent može inicirati 'active open' kako bi započeo slijed od 3 koraka da bi se uspostavila veza. Klijent tada šalje poslužitelju prvi specijalni segment. Poslužitelj odgovara drugim specijalnim TCP segmentom i konačno klijent odgovara trećim specijalnim segmentom. U nastavku slijedi objašnjenje spomenutih segmenata:

1. Prvo klijent šalje specijalni TCP segment poslužitelju. Taj specijalni segment ne sadrži podatke aplikacijske razine. Ima jedan od bitova zastavica u zaglavlju segmenta. To je tzv. SYN bit, postavljen na 1. Iz tog razloga, taj specijalni segment zove se SYN segment. Nadalje, klijent odabire inicijalni redni broj (client_isn) i stavlja ga u polje za redni broj inicijalnog TCP SYN segmenta. Taj segment je uhvaćen u IP datagramu i poslan poslužitelju.

2. Pod pretpostavkom da IP datagram koji sadrži TCP SYN segment stigne do poslužitelja on izdvaja TCP SYN segment iz datagrama, alocira TCP spremnik i varijable i šalje segment kojim odobrava uspostavu veze klijentu. Taj segment odobravanja veze također ne sadrži podatke aplikacijske razine, ali sadrži tri važne informacije u zaglavlju segmenta. Prvo, SYN bit je postavljen na 1. Drugo, Acknowledgment polje zaglavlja TCP segmenta se namješta na isn+1. Na kraju, poslužitelj odabire svoj inicijalni redni broj (server_isn) i stavlja vrijednost u polje zaglavlja TCP segmenta.

3. Kada klijent primi segment odobravanja veze, također alocira spremnik i varijable u vezi. Klijent tada šalje poslužitelju još jedan segment koji potvrđuje da je dobio segment odobravanja veze. To radi tako da stavi vrijednost server_isn+1 u acknowledgement polje zaglavlja. SYN bit postavlja se u 0 budući da je veza uspostavljena.

Kada se sva tri koraka obave, klijent i poslužitelj jedan drugome mogu slati segmente koji sadrže podatke. U svakom budućem segmentu SYN će biti postavljen na 0. Slika 25. prikazuje "three-way handshake" proceduru. [4]



Slika 25. Protokol trostukog rukovanja

4.2. Komunikacija između računala i kontrolera robota

Komunikacija između računala (klijent) i UR5 industrijskog robota (poslužilac) je ostvarena pomoću Ethernet kabla korištenjem TCP komunikacijskog protokola (eng. server-client). Na kontroleru robota ima mnogo aktivnih servera, ali samo jedan ima RealTime komunikaciju, odnosno da aktivno, frekvencijom od 125Hz, šalje sve informacije o trenutnom stanju robota. Taj server se nalazi na TCP portu 30003. RealTime komunikacija omogućava potpunu kontrolu nad svakim segmentom direktno iz C++ skripte.

Server, tj. kontroler robota šalje niz od 756 bajtova koje je potrebno pravilno pročitati kako bi se dobila prava informacija. Svaki bajt je niz od 8 bitova vrijednosti 0 ili 1 (Boolean). Kada taj niz bitova prebacimo u decimalni zapis dobijemo traženu vrijednost, npr. silu, položaj ili temperaturu. Veličina niza koju kontroler šalje ovisi o verziji software-a.

The realtime (125 Hz) communications interface

The realtime communications interface (also known as the Matlab interface) is found at TCP port 30003.

Meaning	Type	Number of values	Size in bytes	Gnuplot col.	Notes
Message Size	integer	1	4		Total message length in bytes
Time	double	1	8	1	Time elapsed since the controller was started
q target	double	6	48	2 - 7	Target joint positions
qd target	double	6	48	8 - 13	Target joint velocities
qdd target	double	6	48	14 - 19	Target joint accelerations
I target	double	6	48	20 - 25	Target joint currents
M target	double	6	48	26 - 31	Target joint moments (torques)
q actual	double	6	48	32 - 37	Actual joint positions
qd actual	double	6	48	38 - 43	Actual joint velocities
I actual	double	6	48	44 - 49	Actual joint currents
Tool Accelerometer values	double	3	24	50 - 53	Tool x,y and z accelerometer values (software version 1.7)
Unused	double	15	120	54 - 67	Unused
TCP force	double	6	48	68 - 73	Generalised forces in the TCP
Tool vector	double	6	48	74 - 79	Cartesian coordinates of the tool: (x,y,z,rx,ry,rz), where rx, ry and rz is a rotation vector representation of the tool orientation
TCP speed	double	6	48	80 - 85	Speed of the tool given in cartesian coordinates
Digital input bits	uint64_t	1	8	86	Current state of the digital inputs. NOTE: these are bits encoded as int64_t!
Motor temperatures	double	6	48	87 - 92	Temperature of each joint in degrees celcius
Controller Timer	double	1	8	93	Controller realtime thread execution time
Test value	double	1	8	94	A value used by Universal Robots software only
Robot Mode	double	1	8	95	Robot control mode (see PolyScopeProgramServer)
TOTAL		**96**	**764**		

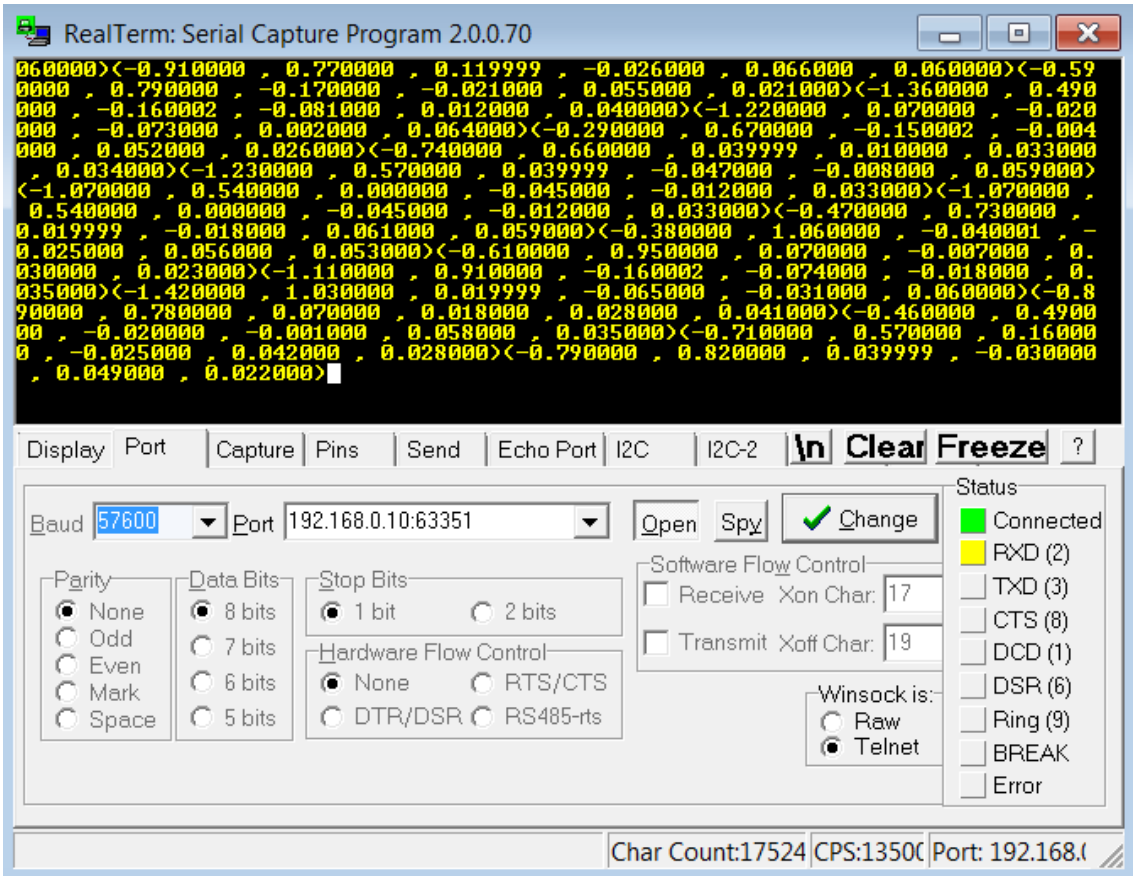
If it is experienced that less than 756 bytes are received, the protocol for the actual received bytes also follows the structure listed above, only not containing the entries at leading up the 756th byte.

Slika 26. Poruka koju šalje kontroler robota

4.3. Komunikacija između računala i senzora

Senzor sile FT 150 spojen na robota šalje string zapise (*hr*. znakovni niz) koji su odvojeni delimiterima, a sastoje se od 6 double varijabli (Double varijabla se koristi za spremanje velikih vrijednosti s pomičnim zarezom, duljine 64bita, opsega $-1.7 \cdot 10^{-308}$ do $1.7 \cdot 10^{308}$). Da bi se pravilno pročitala poruka potrebno je detektirati delimitere. Od velike je važnosti detektirati oba delimitera kako bi sa sigurnošću očitali cijeli znakovni niz u toj poruci, a ne dio, što bi dovelo do pogrešnog očitanja te uzorokovalo velika odstupanja prilikom rada.

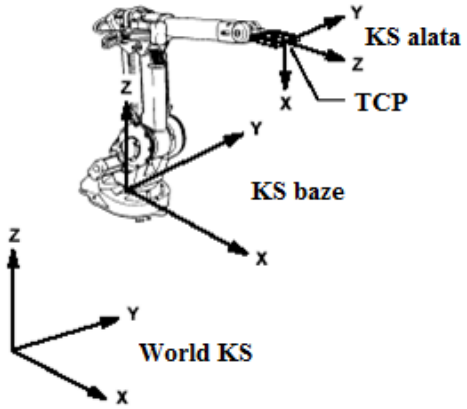
Pomoću RealTerm terminal programa (koji ima mogućnost prilagodbe ispisa i ispravljanja pogrešaka nastalih prilikom komunikacije korisnik-poslužitelj) spajamo se na port 63551 (port namjenjen isključivo za čitanje podataka sa senzora) i čitamo poruku, odnosno niz znakova koje je potrebno pravilno pročitati. Kada je ustanovljen poredak delimitera (znakova koji dijele poruku na potrebne sekcije), jednostavno su pročitani podaci u C++ skripti i prilagođeni za korištenje u aplikaciji.



Slika 27. Prikaz izlaza senzora na RealTerm terminalnom programu

5. KALIBRACIJA

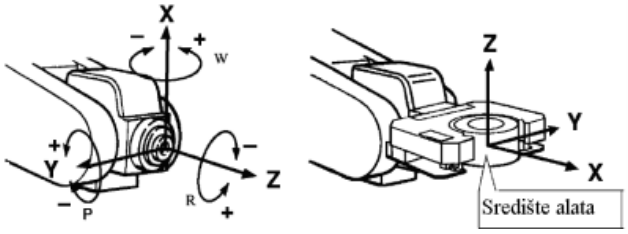
Položaj robota u prostoru definiran je pomoću pozicije i orijentacije, a pošto se na kraju robota nalazi alat koji robot koristi u svom zadatku, orijentacija se odnosi na položaj samog alata u jednom od koordinatnih sustava robota. Manipulator koji ima šest zglobova može dovesti alat u bilo koju poziciju i orijentaciju unutar radnog prostora robota. Ukoliko manipulator ima više od šest zglobova on je reduntantan, odnosno može postići istu poziciju i orijentaciju alata na više načina, a ta sposobnost se koristi za izbjegavanje prepreka u radnom prostoru.



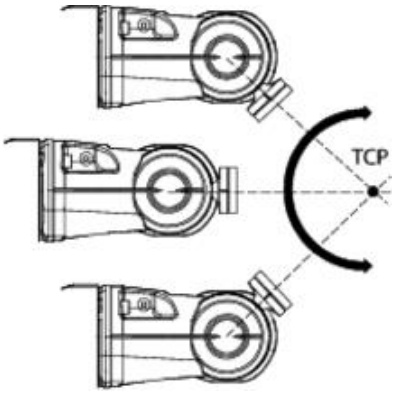
Slika 28. Prikaz koordinatnih sustava

Koordinatni sustav je sustav u kojemu se položaj robota prikazuje koordinatama, i to prostornom translacijom po osi x, y, z, te nizom rotacija oko tih triju osi rx, ry i rz. Zadani koordinatni sustav je kartezijski, pravokutni, desnokretni koordinatni sustav. Pošto je gibanje robota prikazano koordinatama u koordinatnom sustavu baze (eng. *base*), a preostali koordinatni sustavi su definirani u odnosu na isti, potrebno je izračunati matricu transformacije koja će transformirati željeno gibanje alata u koordinatni sustav baze.

Kako robot ima mogućnost korištenja neograničenog broja hvataljki, potrebno je za svaku definirati njen TCP (Tool Center Point), položaj i orijentaciju vrha alata kako bi alat uvijek izvršavao željeno gibanje. [12]



Slika 29. Točka prihvata alata i koordinatni sustav središta alata



Slika 30. Gibanje alata oko definirane radne točke

Kako bi se pravilno proveo postupak bušenja od velike je važnosti pravilno definirati gibaje robota u prostoru. Sustav za bušenje ne sačinjava vizijski sustav, čime bi se uvelike olakšala kalibracija vrha alata i osi gibanja alata, pa je potrebno provesti posebne metode kalibracije, opisane u narednim poglavljima, ali da bi u potpunosti razumjeli postupak, bitno je razumjeti načine zapisa orijentacije.

Orijentacija alata robota zapisana je u obliku trodimenzionalnog vektora, tzv. vektora rotacije. To se naziva Angle-Axis notacija.

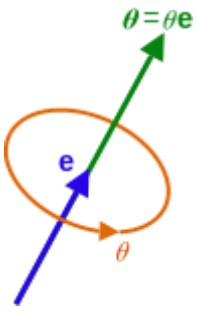
$$\mathbf{V}_{\mathrm{rot}} = [\mathrm{R_X}\ \mathrm{R_Y}\ \mathrm{R_Z}] \tag{1}$$

gdje $\mathrm{R_X}, \mathrm{R_Y}$ i $\mathrm{R_Z}$ predstavljaju kuteve rotacije oko određene osi, u radijanima. Angle-Axis zapis je zapravo način interpretiranja rotacije, gdje je vektor rotacije umnožak jediničnog vektora ê i kuta rotacije θ.

$$\boldsymbol{V}_A = [e_X\ e_Y\ e_Z\ \theta] \tag{2}$$

$$\theta = \sqrt{{R_X}^2 + {R_Y}^2 + {R_Z}^2} \tag{3}$$

$$e_x = \frac{R_X}{\theta}, \qquad e_y = \frac{R_y}{\theta}, \qquad e_z = \frac{R_z}{\theta}; \tag{4}$$



Slika 31. Angle-Axis interpretacija rotacije

Rotacijska DCM matrica (eng. *direction cosine matrix*) zove se i kosinusna jer kosinus kuta između rotiranih jediničnih vektora i pripadnih izvornih određuje (neke) elemente matrice. Svojstva ove matrice su da ima realne parametre, ortogonalna je, determinata joj je 1, a svojstveni vektor čija je pripadna svojstvena vrijednost 1 određuje os rotacije. Rotacijska matrica se može jednostavno izračunati pomoću formule: [13]

$$\boldsymbol{R} = \begin{bmatrix} C * \boldsymbol{e_x}^2 + \boldsymbol{cos\theta} & C * \boldsymbol{e_x} * \boldsymbol{e_y} - \boldsymbol{e_z} * \boldsymbol{sin\theta} & C * \boldsymbol{e_x} * \boldsymbol{e_z} + \boldsymbol{e_y} * \boldsymbol{sin\theta} \\ C * \boldsymbol{e_x} * \boldsymbol{e_y} + \boldsymbol{e_z} * \boldsymbol{sin\theta} & C * \boldsymbol{e_y}^2 + \boldsymbol{cos\theta} & C * \boldsymbol{e_y} * \boldsymbol{e_z} - \boldsymbol{e_x} * \boldsymbol{sin\theta} \\ C * \boldsymbol{e_x} * \boldsymbol{e_z} - \boldsymbol{e_y} * \boldsymbol{sin\theta} & C * \boldsymbol{e_y} * \boldsymbol{e_z} + \boldsymbol{e_x} * \boldsymbol{sin\theta} & C * \boldsymbol{e_z}^2 + \boldsymbol{cos\theta} \end{bmatrix} \quad (5)$$

gdje je

$$C = 1 - \boldsymbol{cos\theta} \quad (6)$$

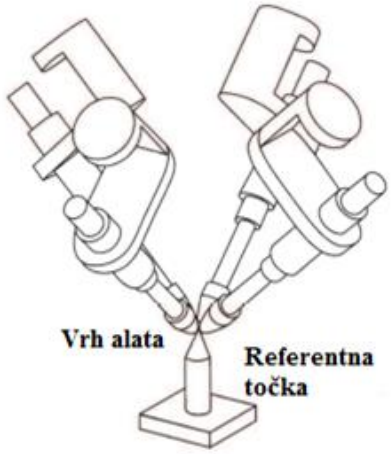
Kako bi se mogle primjeniti navedene jednadžbe u svojem C++ kodu potrebno je instalirati posebnu biblioteku koja se koristi za linearnu algebru, za računanje sa matricama i vektorima, Eigen. Eigen je napredna (eng. *high level*) biblioteka koja uvelike olakšava i omogućava računanje sa matricama i vektorima. Da bi bilo moguće razviti upravljačku C++ aplikaciju za UR5 robota, bilo je potrebno preuzeti sa interneta na računalo, te ju uključiti u zaglavlje skripte (eng. *header*) i dodati pohranjenu lokaciju u svojstva izbornika (eng. *properties*).

```
#include <iostream>
#include <Eigen/Dense>

using Eigen::MatrixXd;

int main()
{
  MatrixXd m(2,2);
  m(0,0) = 3;
  m(1,0) = 2.5;
  m(0,1) = -1;
  m(1,1) = m(1,0) + m(0,1);
  std::cout << m << std::endl;
}
```

Slika 32. Primjer Eigen koda

5.1. Kalibracija vrha alata

Kako bi se odredio vrh alata primjenjuje se metoda pomoću referentne točke. U C++ skripti napisan je program koji je prethodno definiran i testiran u MATLAB programskom paketu. MATLAB je programski jezik visoke razine i interaktivna je okolina za numeričko i matrično računanje, te za vizualizaciju i programiranje. Metoda kalibracije se sastoji od dovođenja vrha alata u istu, referentnu točku, što više puta, ali tako da je svaki put alat pod drugom orijentacijom naspram te točke. Također, minimalan broj točaka za uspješno definiranje jest 3.



Slika 33. Metoda kalibracije vrha alata

Prije početka kalibriranja alata potrebno je negdje u radnom prostoru robota definirati referentnu točku. Ta točka u ovom primjeru je posebno izrađeni element koji ima vrlo preciran i šiljast vrh, u svrhu što točnije kalibracije. Prilikom svakog dovođenja vrha alata u referentnu točku, pohranjuje se vektor rotacije i vektor orijentacije alata, u koordinatnom sustavu baze, u posebnu varijablu. To se ponavlja n puta, te kako je prije navedeno, minimalno 3. Što je veći broj ponavljanja, to je veća točnost. Kada dobijemo željeni broj ponavljanja, pomoću formule (x) dobijemo točne x, y, z udaljenosti vrha alata.

$$X_b = R_1 * X_t + T_1 = R_2 * X_t + T_2 = \cdots = R_N * X_t + T_n \tag{7}$$

Izjednačavanjem navedenih jednadžbi dobijemo matrice pomoću kojih dobijemo vektor X_t, vektor položaja vrha alata, u formi AX=B (x).

$$\begin{bmatrix} R_2 - R_1 \\ R_3 - R_2 \\ \vdots \\ R_n - R_{n-1} \end{bmatrix} X_t = \begin{bmatrix} T_1 - T_2 \\ T_2 - T_3 \\ \vdots \\ T_{n-1} - T_n \end{bmatrix} \tag{8}$$

Matrica R je rotacijska matrica trenutne orijentacije alata, a matrica T je matrica trenutnog položaja alata u base koordinatnom sustavu robota. Ukoliko A i B matrice nisu singularne (determinante tih matrica su različite od nule), X_t (matricu položaja vrha alata u odnosu na TCP robota) dobijemo pomoću sljedeće jednadžbe:

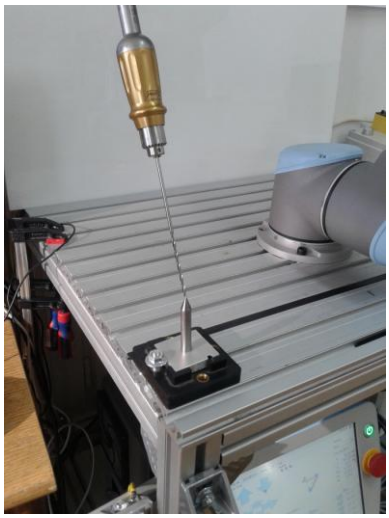
$$X_t = (A^T * A)^{-1} A^T * B \tag{9}$$

gdje su A i B:

$$A = \begin{bmatrix} R_2 - R_1 \\ R_3 - R_2 \\ \vdots \\ R_n - R_{n-1} \end{bmatrix}, B = \begin{bmatrix} T_1 - T_2 \\ T_2 - T_3 \\ \vdots \\ T_{n-1} - T_n \end{bmatrix}. \tag{10}$$

```
Program je aktivan.
Molim odaberite funkciju:
1. Kalibracija TCP-a
2. Izlaz
1
Molim postavite robota u PRVU poziciju i pritisnite enter.

R_1
  0.0143282    -0.996044  -0.0876953
   -0.98647 0.000243664   -0.163943
   0.163316    0.0888578   -0.982564
 0.206393
-0.332471
 0.304842
Molim postavite robota u DRUGU poziciju i pritisnite enter.

R_2
 0.945495 -0.193846 -0.261654
 -0.26224 -0.929624 -0.258899
-0.193853  0.313404  -0.92979
 0.287729
-0.270671
 0.273651
Molim postavite robota u TRECU poziciju i pritisnite enter.

R_3
-0.733795 -0.553849  0.393441
-0.678617  0.570271 -0.462893
0.0320051 -0.606665 -0.794313
0.0663424
-0.256028
 0.259107
Kalibracija je zavrsena.
-0.0516652
0.00396131
  0.208284
Molim odaberite funkciju:
1. Kalibracija TCP-a
2. Izlaz
```

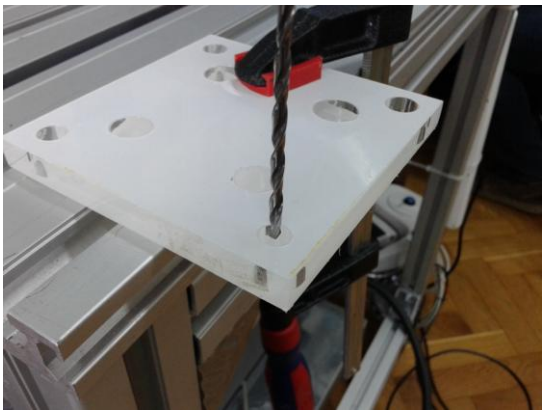
Slika 34. Ispis konzole nakon kalibracije vrha alata



Slika 35. Vrh bušilice u referentnoj točki

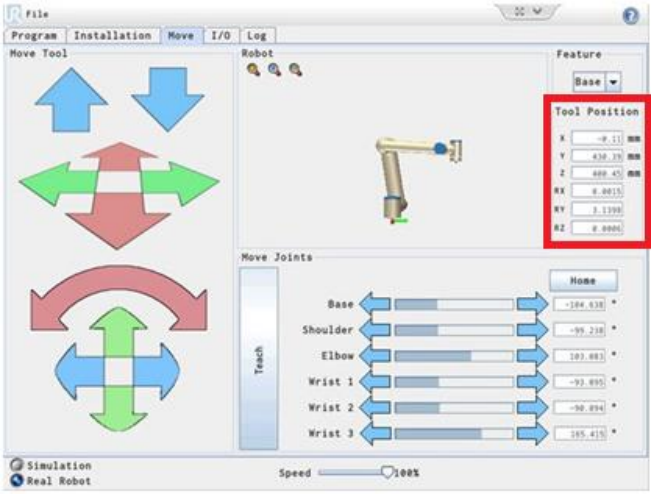
5.2. Kalibracija orijentacije alata

Kalibracija orijentacije alata bez vizijskog sustava zahtjeva više vremena i vještine kako bi se postigla ista mjera preciznosti. Za pronalazak vektora orijentacije alata potrebno je prvo definirati os po kojoj se giba alat. Za početak definira se linija (eng. *line*) u Feature opciji Polyscope korisničkog sučelja. Za što točniju definiciju linije, odnosno osi gibanja alata, kao pomagalo koristio se provrt u pločici se metodom iteracije tražilo što točnije poklapanje osi gibanja alata sa osi provrta. Nakon što je postignuto poklapanje osi, pomoću dvije točke na udaljenim krajevima iste, definira se linija.



Slika 36. Definiranje osi gibanja alata

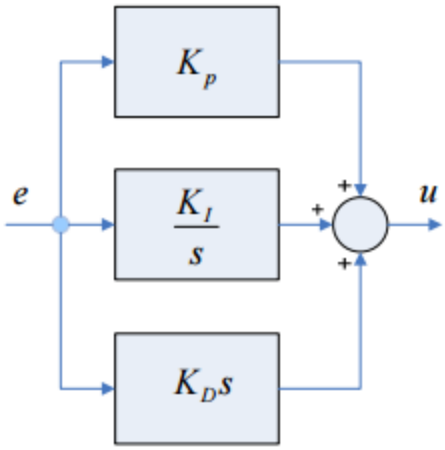
Nakon što se definirala os gibanja bušilice, očitavaju se vrijednosti vektora rotacije u base koordinatnom sustavu za TCP i definiranu Feature liniju. Dobivene vrijednosti unesu se u MATLAB kod. Za izračun kuteva orijentacije bušilice, potrebno je vektorski zapis prebaciti u matricu rotacije (DCM zapis). Jednostavnim množenjem inverzne matrice orijentacije alata sa matricom rotacije osi gibanja bušilice dobije se DCM matrični zapis iz kojeg se, ponovnom pretvorbom u vektorski zapis, očitaju kutevi rotacije. Metodom iteracije, tj. ponavljanjem cijelog postupka povećava se točnost vektora rotacije osi alata.



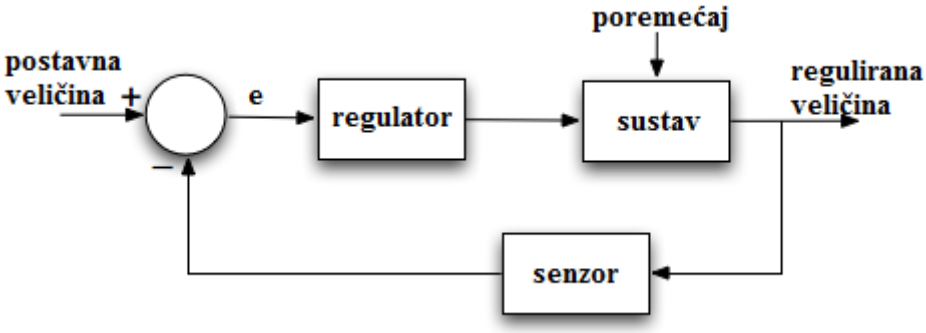
Slika 37. Očitanje kuteva u Polyscope korisničkom sučelju

6. SIMULACIJSKI MODEL BUŠENJA KOSTI

Matematički model opisuje sustav i njegovo ponašanje za zadane parametre. Model opisuje sustav pomoću varijabli (parametara, svojstava) i odnose među njima koji su definirani matematičkim relacijama. Koristi se za predviđanje ponašanja sustava u kritičnim uvjetima i situacijama koje su od ključne važnosti (kao u trenutknu proboja bušilice u i kroz kost). Također, služi za određivanje parametara bušenja i promatranja samog procesa u istima. Kada se definiraju parametri bušenja, koji su preuzeti iz realnih slučaja, simulira se sustav, re se prati njegovo ponašanje. Kako bi se postiglo željeno ponašanje sustava potrebno je uvesti regulaciju. Postizanje željene brzine tokom bušenja i stabilnost sustava postiže se pravilnim podešavanjem pojačanja regulatora, što će biti prikazano u narednom poglavlju. Važan dio matematičkog modela je validacija ili procjena modela, kojom se provjerava da li model opisuje sustav točno ili ne, a najbolji način za to je upravo provjera poklapanja simulacijskih i eksperimentalnih rezultata.

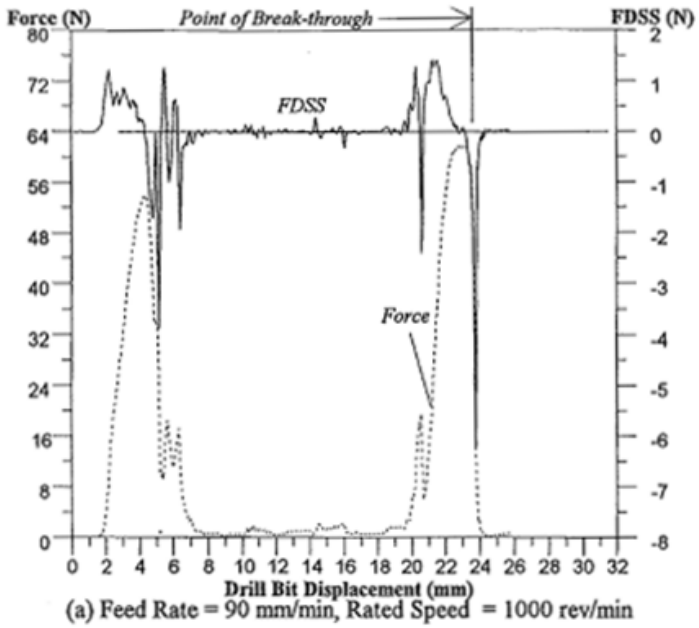


Slika 38. Osnovna forma / blok struktura PID regulatora

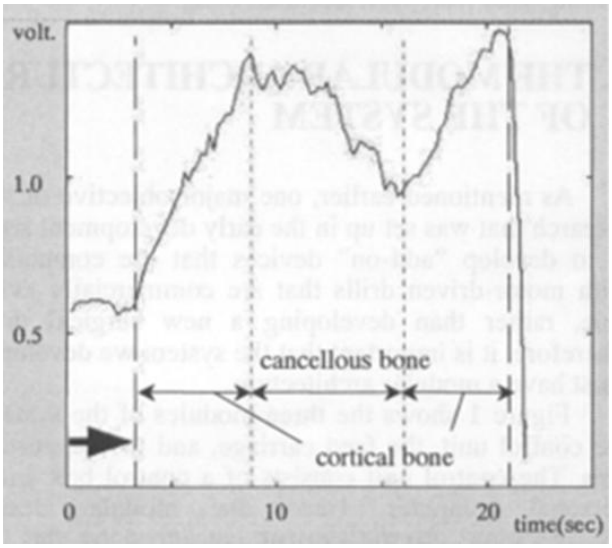


Slika 39. Simulacijski model

Za potrebe simulacije modela korišten je MATLAB programski paket. Cijeli kod je zapisan u obliku skripte, te samom promjenom parametara uočavaju se odgovarajući simulacijski odzivi. Ponavljanjem simulacija uz promjenu parametara traže se najpovoljnije vrijednosti pojačanja koje se primjeniti u realnom modelu. Vrijednosti parametara korištenih u simulaciji očitani su iz literature [14].



Slika 40. Karakteristične sile i brzine vrtnje kod bušenja kosti



Slika 41. Odziv električne bušilice prilikom bušenja kosti

6.1. Izrada simulacijskog modela

Dinamika bušenja je opisana translacijskom dinamikom

$$m\ddot{x} = F_x(x, \omega) + F_P \tag{11}$$

koja, prenesna u prostor stanja:

$$\dot{x} = v \tag{12}$$

$$\dot{v} = \frac{1}{m} F_x(x, \omega) + \frac{1}{m} F_P \tag{13}$$

Kost u simulacijskom modelu je predstavljena kao sila otpora. U prvom primjeru korištena je homogena kost koja je prikazana kao pravokutna barijera širine jednake debljini kosti i visine koja je (obrnuto) proporcionalna rotacijskoj brzini bušilice. Pretpostavlja se da je ta obrnuta proporcionalnost modelirana opadajućom eksponencijalnom funkcijom. U nastavku homogena kost je zamjenjena realnim modelom kosti koji je preslika odziva električne bušilice prilikom bušenja kosti, tj. realnijim otporom prilikom bušenja.

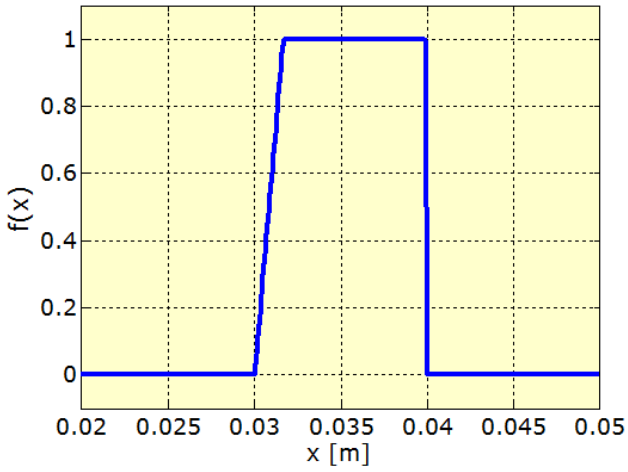
Parametri:

- F_p - translacijska sila pritiska bušilice na kost
- F_0 - visina barijere
- x_0 - početna pozicija
- v_0 - početna brzina
- m - masa
- k - faktor opadanja
- v_d - željena translacijska brzina
- omega - rotacijska brzina bušilice
- K - nagib funkcije barijere

Konačna funkcija:

$$F_x(x,\omega) = -F_0 e^{-k\omega} f(x) sign(\dot{x}(0)) \tag{14}$$

Gdje je F_0 sila barijere koja je potrebna da se probije kost (bez rotacijske brzine), koja je veća od sile pritiska F_p. Kako rotacijska brzina raste, tako visina barijere opada, a parametar k definira 'brzinu' tog opadanja. Za određenu kritičnu brzinu ω_c visina barijere će se izjednačiti sa silom pritiska $-F_0 e^{-k\omega} = F_p$, tako da je $F_0 e^{-k\omega} < F_p$ za $\omega > \omega_c$. Funkcija predznaka $sign(\dot{x}(0))$ služi samo da smjer sile bude odgovarajući u odnosu na smjer početne brzine $v(0) = \dot{x}(0)$. Rotacijska brzina u ovom modelu je konstantna.



Slika 42. Homogena barijera / kost

Regulator tvori regulacijsko odstupanje e(t) = v - y(t), a podešava se izborom podesivih parametara, tako da se postigne najpovoljnije regulacijsko vladanje sustava. Odabrali smo PI regulator jer nam nije potrebno derivacijsko djelovanje, samo integralno uz proporcionalno da bi postigli željenu brzinu prilikm bušenja kosti.

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de}{dt} \tag{15}$$

gdje su K_p, K_i i K_d pojačanja regulatora, $e(t)$ regulacijska pogreška, a $u(t)$ izlaz iz regulatora.

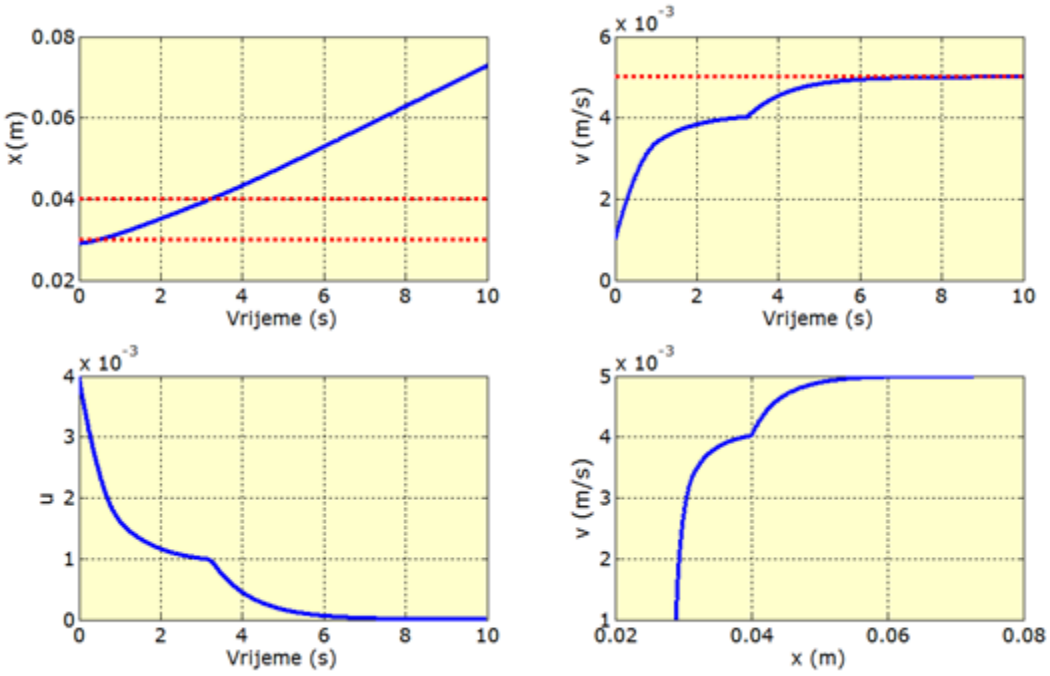
Funkcija ode45

U računalnom programu Matlab korištena je funkcija ode45 (eng. ordinary differential equations). Za numeričko integriranje diferencijalnih jednadžbi, koristimo dvije funkcije: ode23( ), i ode45( ). Ode23( ) koristi par formula drugog i trećeg stupnja dok ode45( ) koristi Runge-Kutta-Fehlberg formule četvrtog i petog stupnja za automatsko koračno integriranje.

```
[t,y] = ode45('ime_dat',[TSPAN],[Y0],options);
```

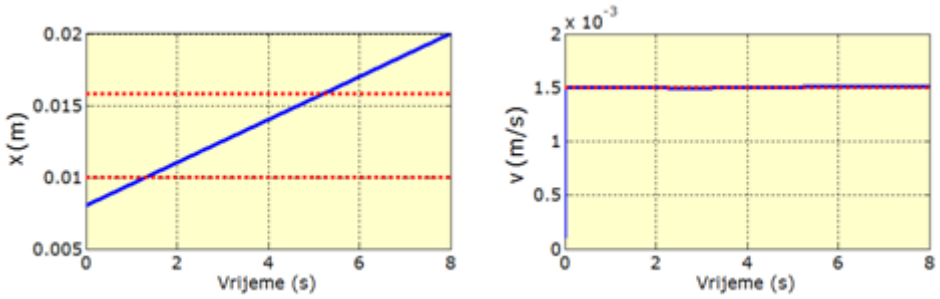
- 'ime_dat' - ime funkcije koju pozivamo
- TSPAN = [T0 TFINAL] - vremenski integral integriranja sustava dif. jed.
- Y0 - početni uvijeti (vrijednosti)
- [t,y] - rezultat, **t** i **y** su vektori i sadrže rješenje za svaki korak integracije

Simulacija



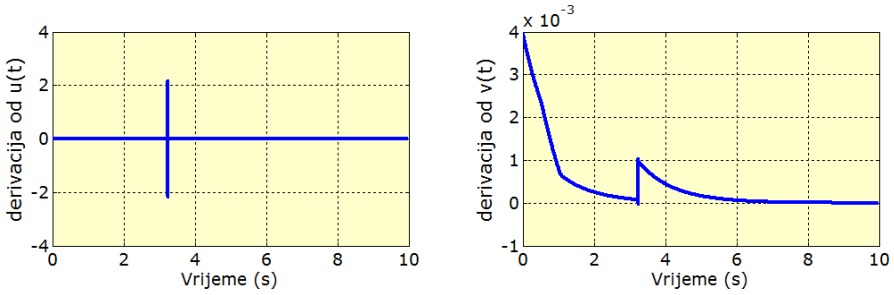
Slika 43. Simulacija bušenja samo sa P regulatorom

Tijekom prolaska svrdla kod kost translacijska brzina ne može postići željenu vrijednost zato što je korišten samo P regulator bez integralnog djelovanja, što je vidljivo na Slici 43. Kako bi se postigla željena translacijska brzina prilikom bušenja kosti dodaje se potrebno integralno djelovanje. Prilikom podešavanja pojačanja regulatora treba paziti, veliki Kp uzrokuje brzu dinamiku, ali je lošija stabilnost.



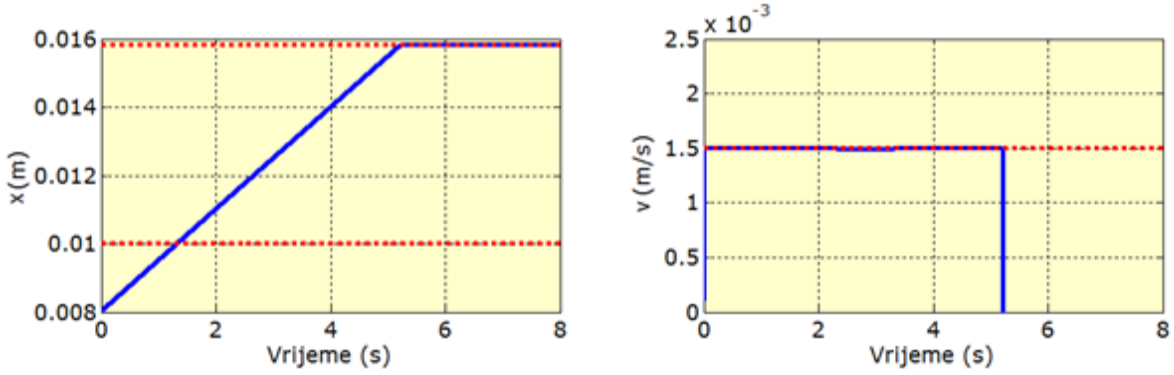
Slika 44. Bušenje kosti sa PI regulatorom

Nakon što se postigla tražena translacijska brzina, potrebno je pravovremeno zaustaviti proces bušenja kako se ne bi oštetio mozak. U trenutku izlaska svdrla iz kosti javlja se diskontinuitet brzine i upravljačke varijable. Kao rezultat derivacije tih varijabli nastaju pomoćne varijable (Slika 44.) koje mogu poslužiti za detekciju trenutka i položaja svrdla prilikom izlaska iz kosti. Derivacija je samo jedna od mogućih metoda detekcije.



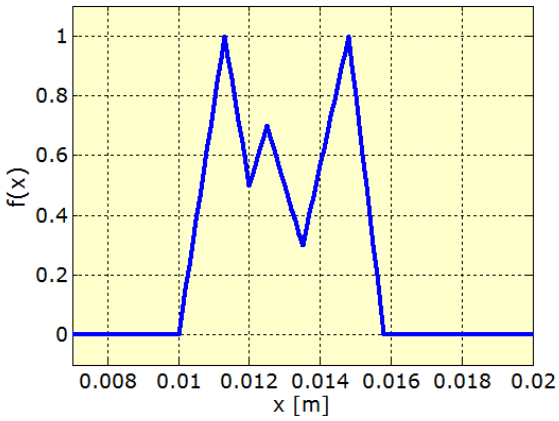
Slika 45. Pomoćne varijable

Threshold je bitan parametar koji se određuje na temelju eksperimenata ili iskustveno. Ukoliko se postavi mali threshold neće biti velikog proboja nakon izlaska iz kosti, ali zato postoji mogućnost da se detektira neki raniji pad, odnosno derivacija, te će bušenje biti prekinuto prije samog kraja. Opet, ukoliko threshold bude prevelik, izbjeći će se mogućnost ranijeg prekida bušenja, ali tada prijeti opasnost od prekasnog prekida nakon izlaska iz kosti i mogućeg oštećenja mozga.

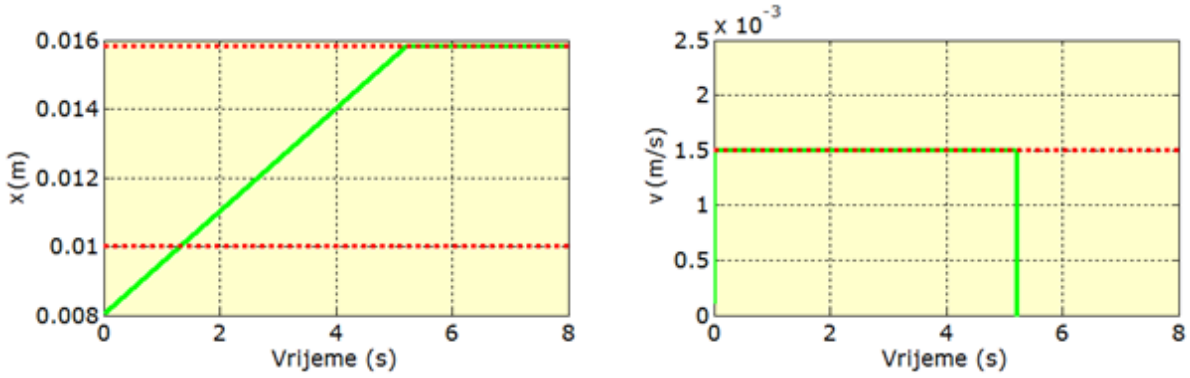


Slika 46. Pravovremeno zaustavljanje procesa bušenja

Nakon postignutih željenih rezultata odziva i pravovremenog zaustavljanja na homogenoj kosti, isti model se primjenjuje na realnijem modelu barijere (Slika 47.). Za novi model barijere ne vrijede prije definirani parametri pojačanja i detekcije proboja, pa se ponavlja simulaciju sve dok ne postignu zadovoljavajući uvijeti. Konačne razultate sa pravilno postavljenim parametrima prikazani su na Slici 48.



Slika 47. Realniji model barijere / otpora kosti

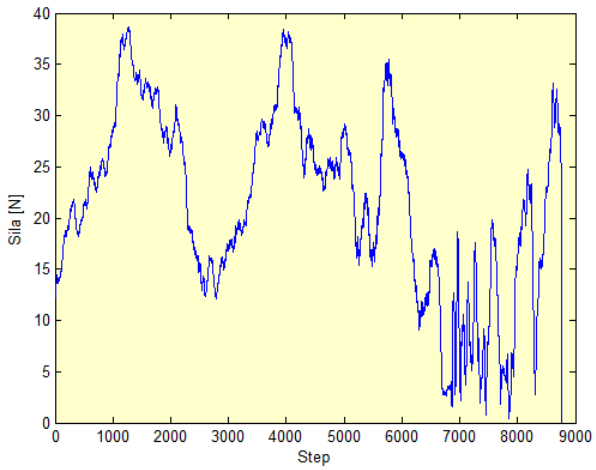


Slika 48. Odziv simulacije bušenja realnijeg modela kosti

7. IMPLEMENTACIJA RAZVIJENOG MODELA VOĐENJA

Nakon simuliranja procesa bušenja u Matlab-u, slijedi bušenje eksperimentalnog nehomogenog modela, sa primjenom načela postavljenih u simulacijskom modeliranju. Implemenatacija PID regulatora u C++ aplikaciji, uz primjenu više istovremenih skripti (eng. *multithreading*) je jako izazovno, prvenstveno jer svaki dio koda može usporiti čitanje podataka ili njihovu obradu, pa je kompiliranje koda od ključne važnosti. Cilj modeliranja je održati konstantnu (referentnu) silu bušenja tokom bušenja kroz nehomogeni materijal, uz očekivane poremećaje, te pravovremeno zaustavljanje prilikom probijanja.

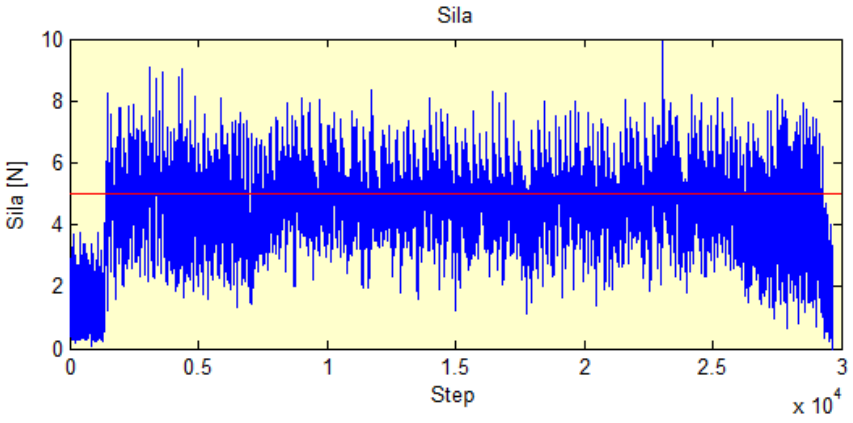
UR5 robotska ruka ima mogućnost mjerenja sile, odnosno mogućnost mjerenja promjene struje koju troše motori uslijed promjene otpora, a koja je proporcionalna sili. Ta metoda nije precizna i uzrokuje velike promjene signala uslijed promjene gibanja smjera robota, što je onemogućilo primjenu tog parametra procesu regulacije sile tokom bušenja.



Slika 49. Velike oscilacije senzora sile UR5 robotke ruke

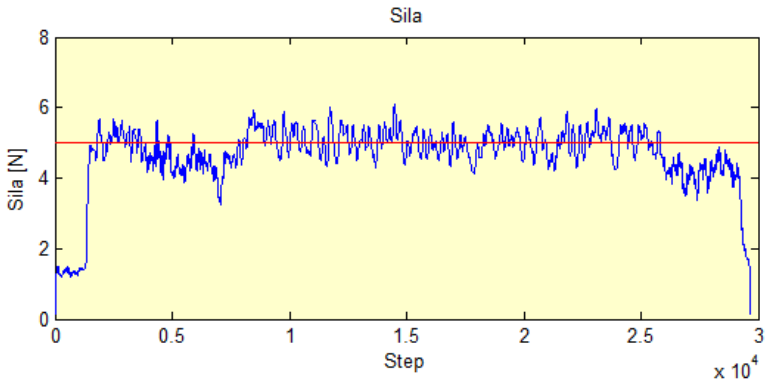
Iako ne možemo iskoristi očitanu silu UR5 robotske ruke, trenutni položaj robota, njegove TCP koordinate koristiti ćemo za bolju kontrolu proboja kosti, jer je debljina kosti unaprijed poznata, a samom detekcijom promjene sile pri kontaktu sa kosti kortikalisa, dobivamo početnu točku bušenja. Sama primjena u regulaciji biti će opisna u narednom poglavlju.

Senzor sile FT 150 daje puno stabilniji izlaz koji ima šum od 0.5N, što je zanemarivo naspram njegovog mjernog opsega od +-150N, ali ne i za radni opseg koji se primjenjuje prilikom bušenja lubanje. Šum, vibracije uzrokovane nesavršenom simetrijom svrdla, njegovom istrošenošću i sl., eliminiraju se primjenom filtera i dobro podešenih parametara PID regulatora bušenja.



Slika 50. Odziv senzora sile FT 150

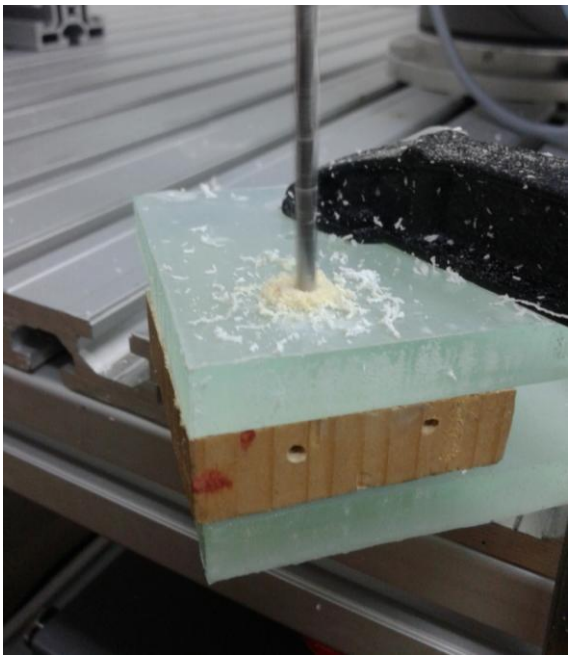
Medijan Filter (eng. / Matlab *medfilt1*) je nelinearan window filter koji lako otklanja šum, dok čuva rubove. Osnovna ideja filtera je za bilo element signala gledati na sve elemente u susjedstvu i očitati element najsličniji drugima. Po svom principu rada sličan je 'mean' ili 'average' filterima. U Matlabu med filter primjenjuje jednodimenzionalni filter trećeg reda (ili n reda) na ulazni vektor, te kao izlaz daje vektor iste duljine. Opisani filter je lako primjeniti u c++ skripti, ali zbog velikog broja petlji i čitanja, zahtjeva dosta vremena i procesorske snage. Primjena filtera tokom bušenja nije se pokazala dobrom jer je usporila sam proces regulacije bušenja. Isto tako, od velike je važnosti detektirati točan trenutak pada signala sile, jer upravo taj pad služi za pravovremeno zaustavljanje procesa bušenja.



Slika 51. Filtrirani signal regulirane sile

Predmet bušenja

Model bušenja za eksprerimentalno bušenje, prije bušenja prave kosti, je kompozit sastavljen od tri sloja, dva sloja polimetil-metakrilat (eng. *poly methyl methacrylate, PMMA*) i jednog međusloja suhog drveta koji imitira mekši sloj kompaktne kosti. Model je složen prema približnim mehaničkim svojstvima kosti. Najbitniji parametar je čvrstoća, mehaničko svojstvo materijala da pruža otpor djelovanju sile. Ponavljanjem bušenja i promatranjem djelovanja pojedinih parametara pojačanja regulatora, možemo predpostaviti iste za bušenje životinjske kosti.



Slika 52. Eksperimentalna nehomogena struktura

Tablica 2. Usporedba mehaničkih svojstava materijala

Materijal	Vlačna čvrstoća (N/mm^2)	Tlačna čvrstoća (N/mm^2)	Modul elastičnosti (kN/mm^2)
Kompaktna kost	50-151	100-230	7-30
Titan	345	250-600	102.7
Nehrđajući čelik	465-950	1000	200
Poli(metil metakrilat)	48-76	83-124	1.8-3.8

Modeliranje PID regulatora u C++ se sastoji od definiranja pojačanja i izračuna regulacijske pogreše koju množimo sa pojačanjem. Pogreška regulacije:

- ep = postavna sila bušenja – stvarna sila bušenja
- ei = ei + ep * dt
- ed = (ep-ep_stari) / dt

Nakon implementacije PID regulatora u c++ skripti, postavljamo pojačanja i eksprerimentalnom metodom određujemo vrijednosti svakog.

Tablica 3. Utjecaji pojedinih parametara PID regulatora

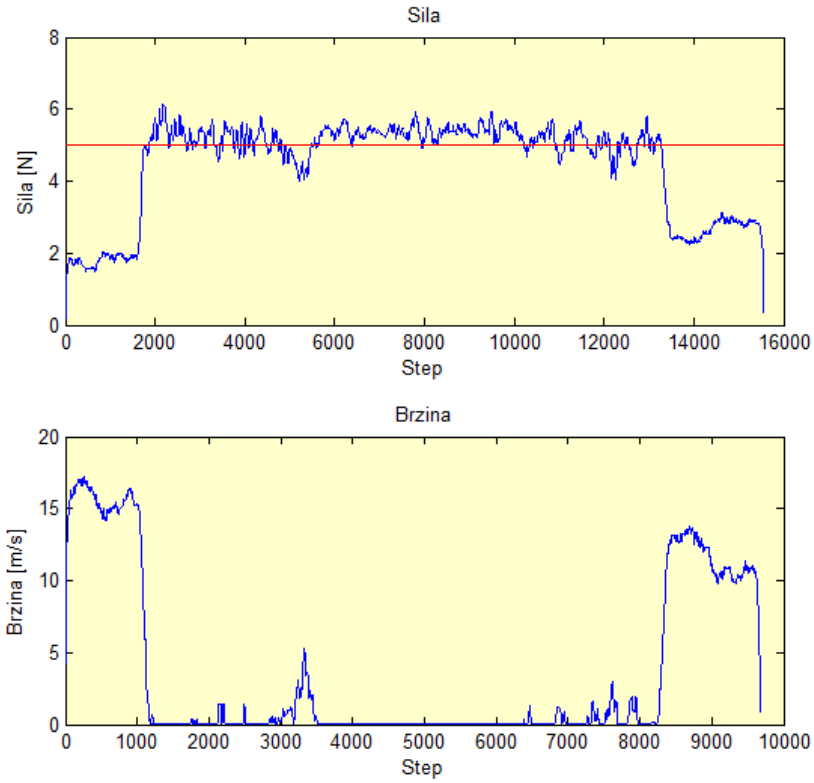
Parametar	Vrijeme porasta	Maksimalni prebačaj	Vrijeme smirivanja	Pogreške u ustaljenom stanju
Kp	smanjuje	povećava	malo utječe	smanjuje, ali ne može potpuno ukloniti
Ki	smanjuje	povećava	povećava	potpuno uklanja
Kd	malo utječe	smanjuje	smanjuje	malo utječe

Postoji više metoda podešavanja PID parametara, poput agresivne Ziegler–Nicholsove, Tyreus Luybenove do jednostavne Cohen–Coonove koja se izvodi offline i idealna je za sustave prvog reda.

Pseudokod za PID regulator:

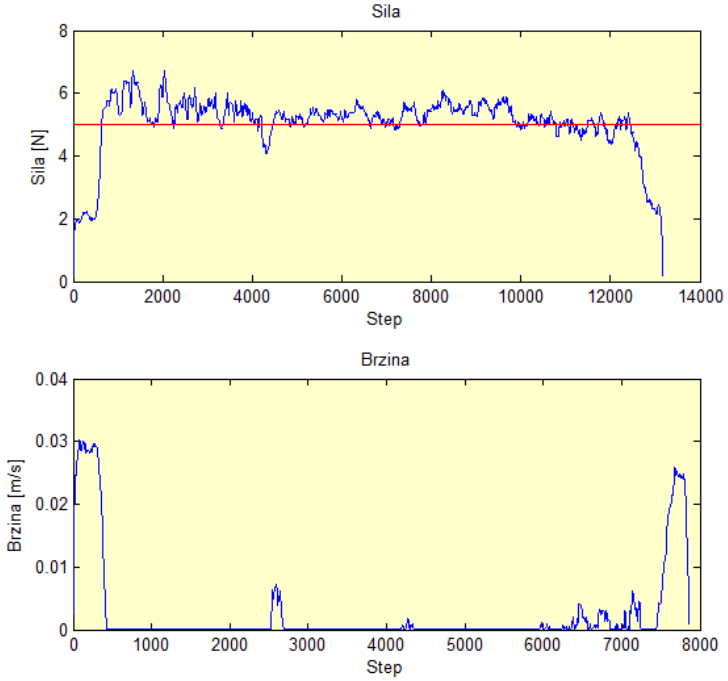
```
greška_st = 0
integral = 0
početak:
  greška = referenca – izmjerena_vrijednost
  integral = integral + greška*dt
  derivacija = (greška - greška_st)/dt
  izlaz = Kp*greška + Ki*integral + Kd*derivacija
  greška_st = greška
  čekaj(dt)
  idi_na početak
```

U prvom slučaju postavljamo malo vrijednost proporcionalnog pojačanja, a ostala isključujemo jednostavnim postavljanjem u nulu. Kao referentnu silu postavljamo 5N. Ukoliko bi stavili premalu silu vrijeme bušenja bi se značajno povećalo, a samim povećanjem vremena, povećava se i razvijena temperatura bušenja koja ima negativan utjecan na proces bušenja i oštećuje predmet bušenja, kost.

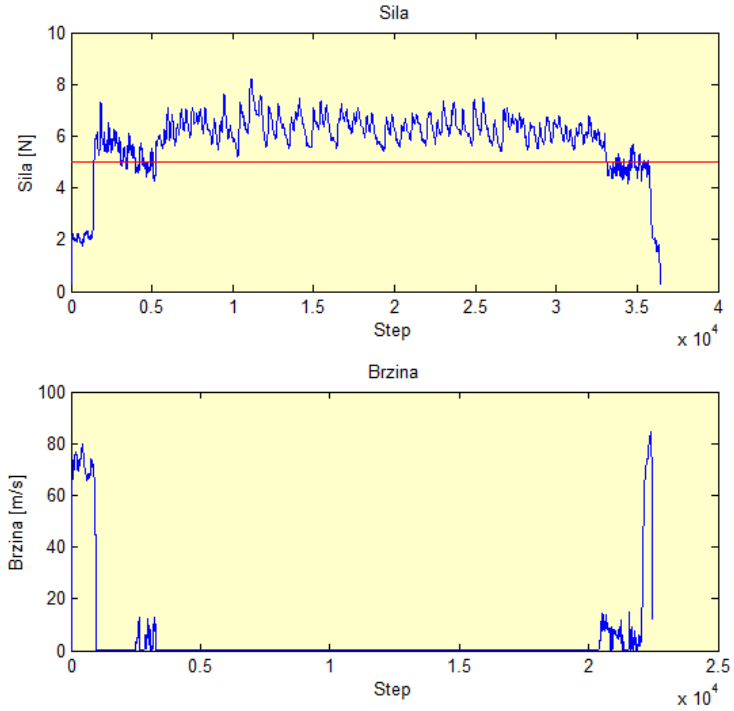


Slika 53. Kp=3 Ki=0

Malo proporcionalo pojačanje sasvim dobro dobro drži konstantnu vrijednost sile kroz promjenu strukture modela. Narednim povećanjem proporcionalnog djelovanja tražimo najbolji mogući odziv, ali do određene mjere. Preveliki iznos pojačanja kao produkt će imati prevelika nadvišenja, a u krajnjoj mjeri i nestabilnost sustava.



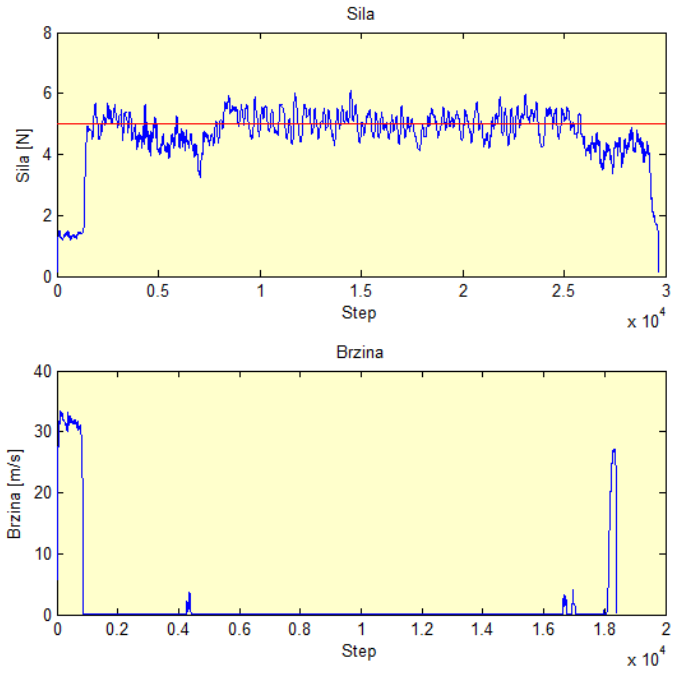
Slika 54. Kp=10 Ki=0



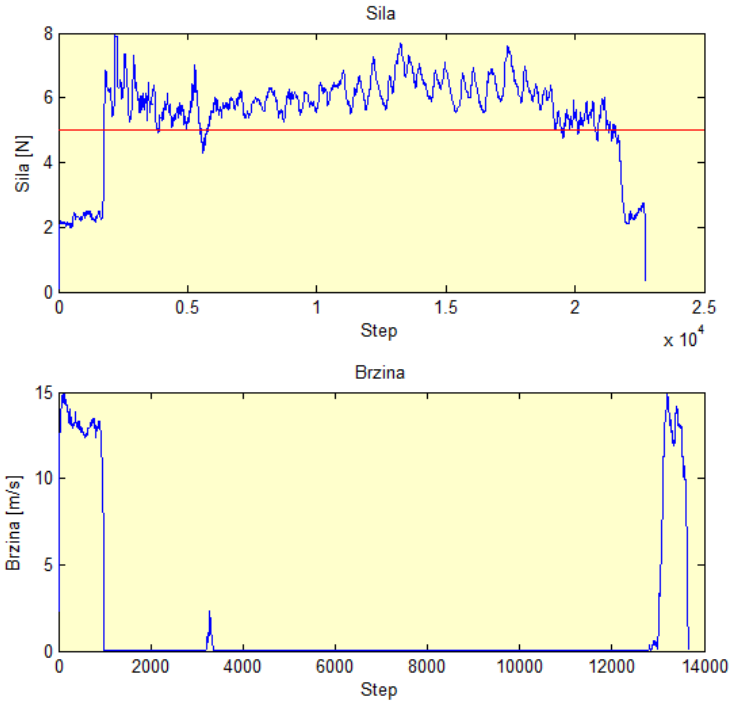
Slika 55. Kp=25 Ki=0

Preveliki iznos proporcionalnog pojačanja za posljedicu ima nadvišenje

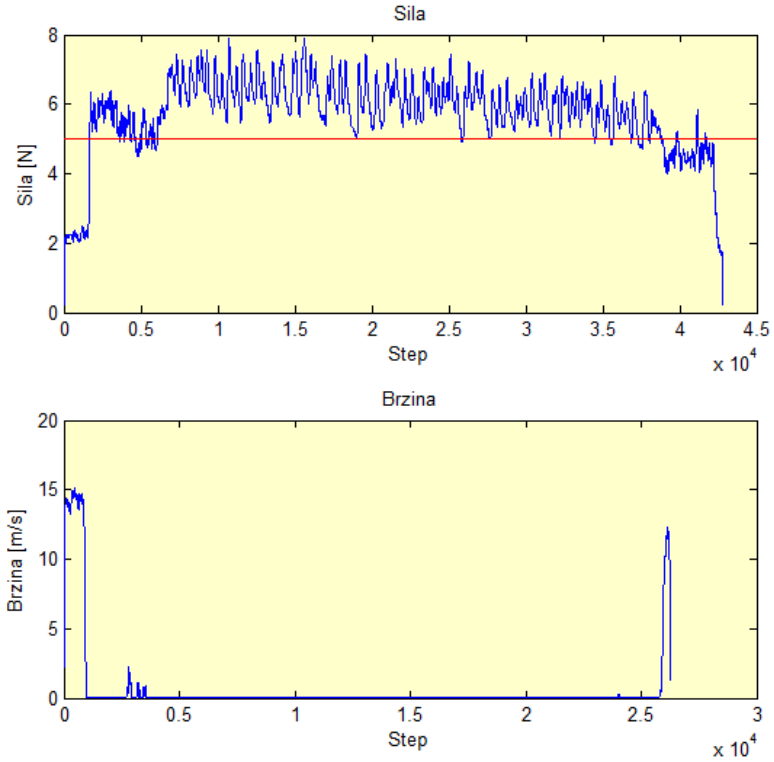
Detekcijom prevelikog proporcionalnog pojačanja, vraćamo prvotnu, stabilniju vrijednost i dodajemo integracijsko djelovanje.



Slika 56. Kp=10 Ki=1



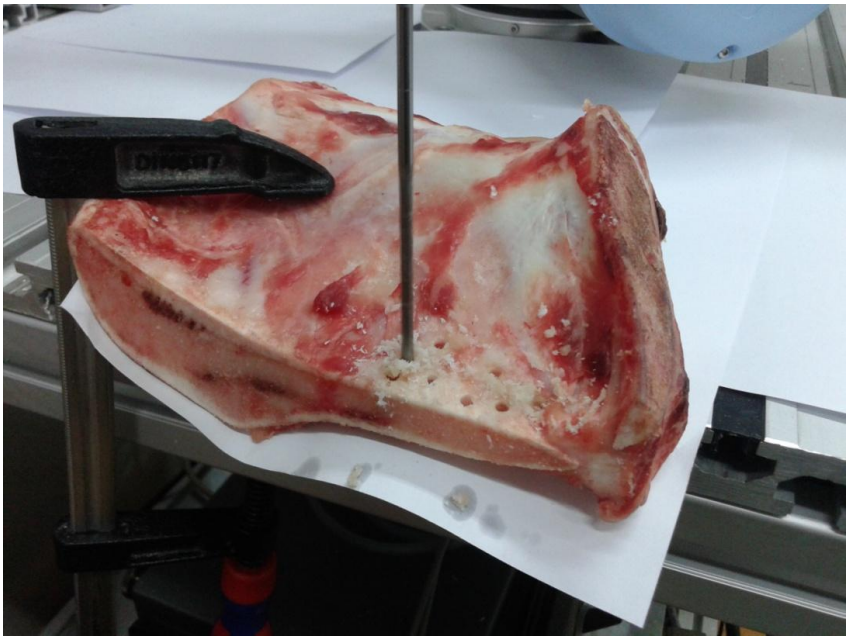
Slika 57. Kp=10 Ki=5



Slika 58. Kp=10 Ki=7

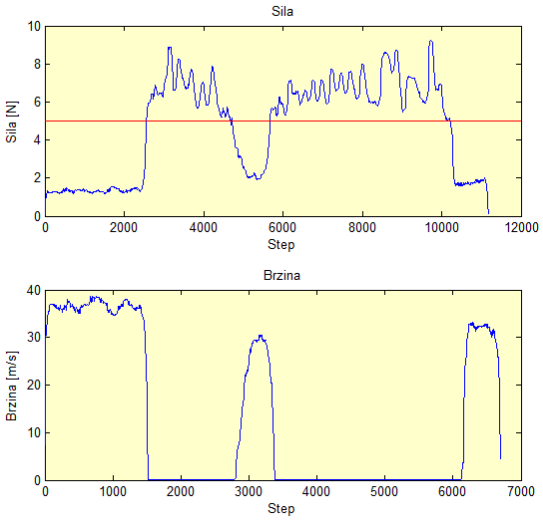
Preveliki iznos integralnog pojačanja za posljedicu ima ogroman prebačaj i veliki šum. Izabiremo parametare iz slučaja najboljeg odziva koje ćemo primjeniti na modelu bušenja stvarne kosti. Mali iznos referentne sile bušenja za posljedicu je imao lagano modeliranje regulatora, a vrijeme bušenja je i dalje ostalo kratko, te zbog toga nije došlo do porasta temperature. U realnom slučaju bušenja uvijek je prisutno istovremeno podmazivanje, u slučaju bušenja lubanje, podmazivanje fiziološkom otopinom.

Bušenje kosti



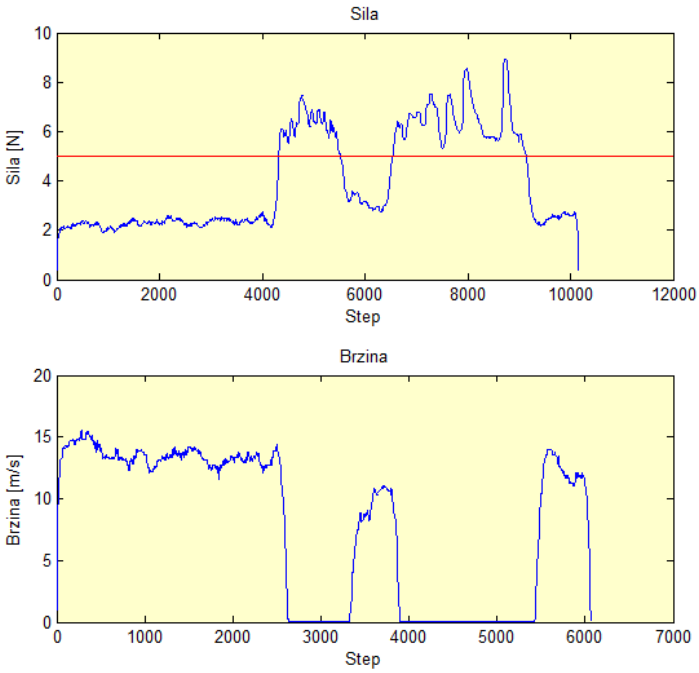
Slika 59. Bušenje životinjske kosti

U prihvat stavljamo komad životinjske kosti koja po svojoj strukturi najsličnije opisuje kost svoda lubanje. Kao što se vidi iz Slika 59. kost se sastoji od 3 sloja, od kojih su gornji i donji puno veće krutosti od spužvastog međusloja. Koristeći najoptimalnije parametre za bušenje kosti dobivene u slučaju bušenja kompozita u prethodnom poglavlju, bušimo kost.

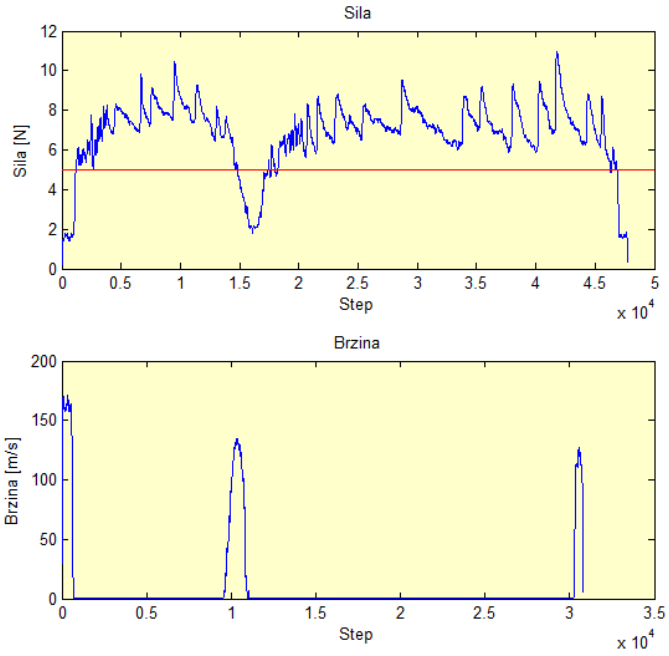


Slika 60. Kp=10 Ki=1

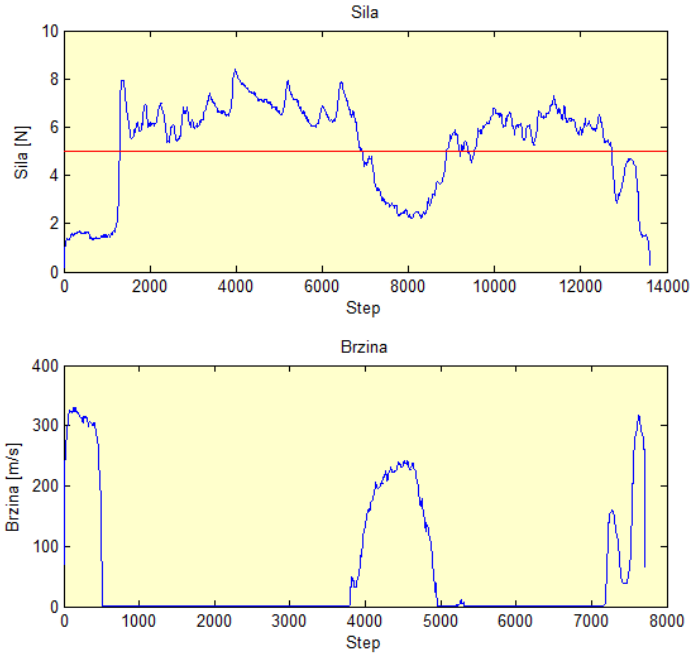
Korišteni parametri su se pokazali nimalo prihvatljivi, uz veliki prebačaj, vrijeme smirivanja i pogrešku. Ponavljamo postupak bušenja uz druge parametre te njihovu optimizaciju.



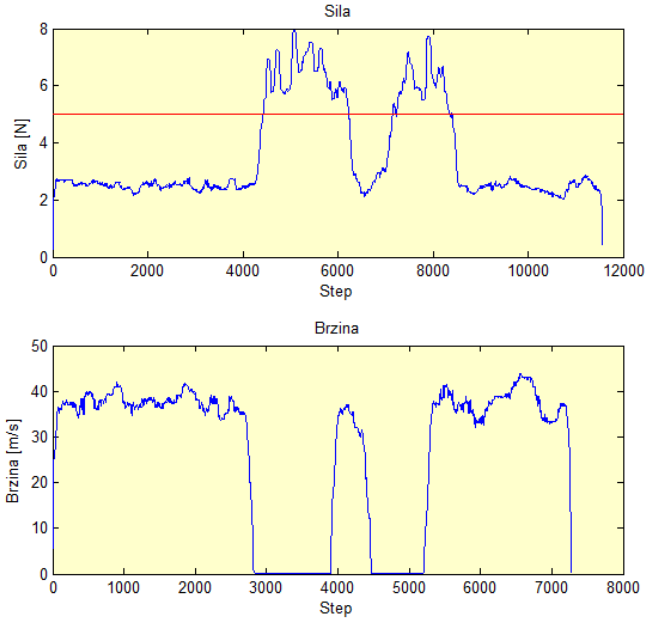
Slika 61. Kp=12 Ki=1.5



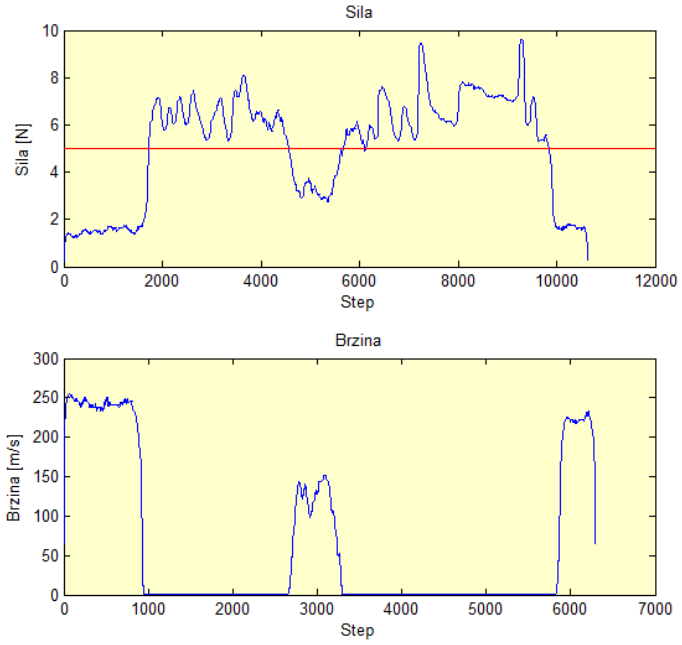
Slika 62. Kp=20 Ki=3



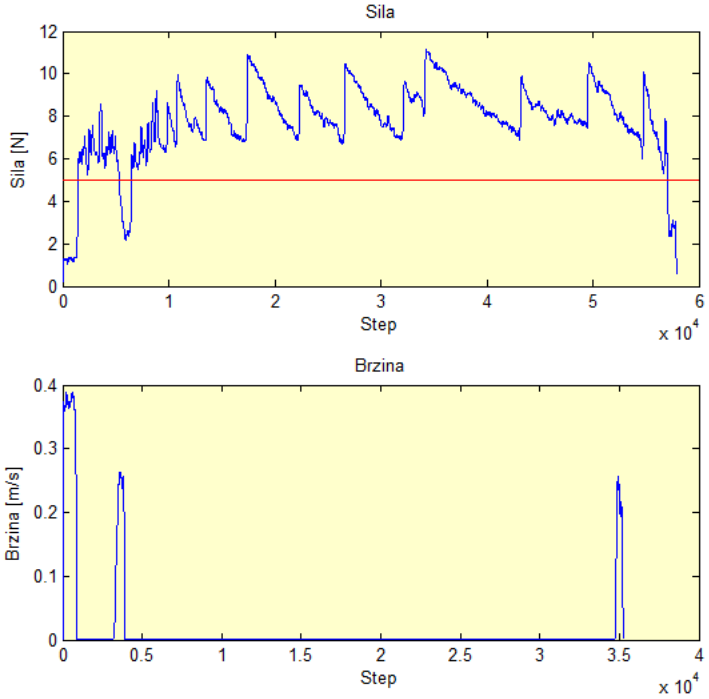
Slika 63. Kp=15 Ki=2 Kd=0.01



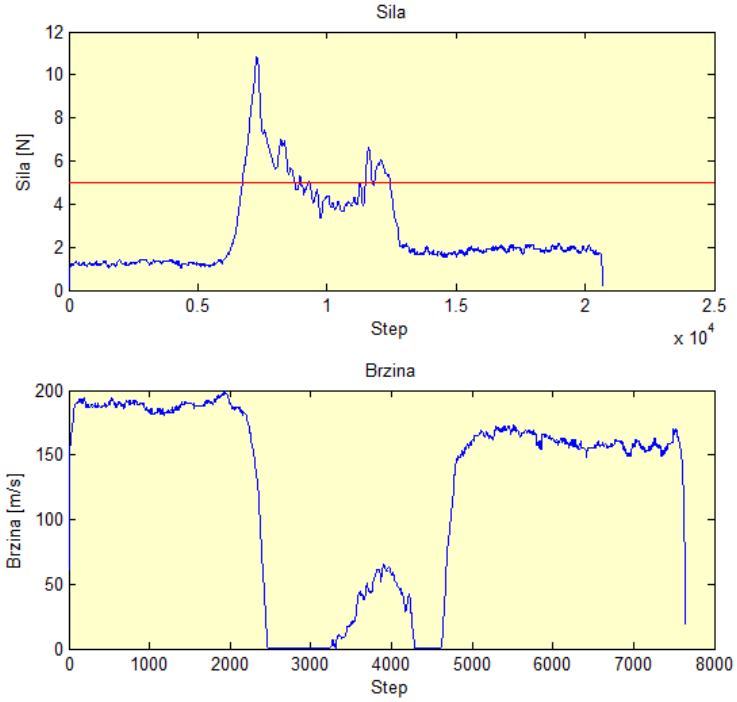
Slika 64. Kp=10 Ki=0.1 Kd=0



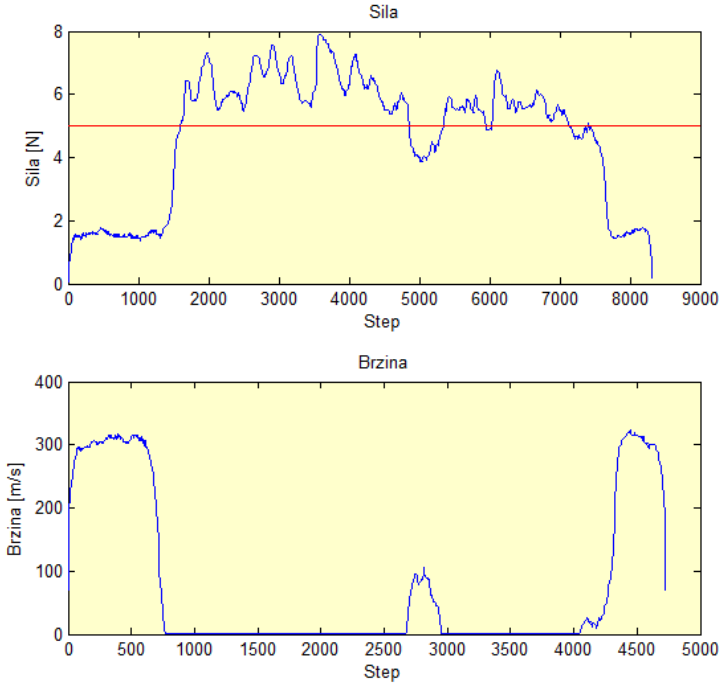
Slika 65. Kp=35 Ki=1 Kd=0.01



Slika 66. Kp=20 Ki=0 Kd=3



Slika 67. Kp=2 Ki=0.01 Kd=0



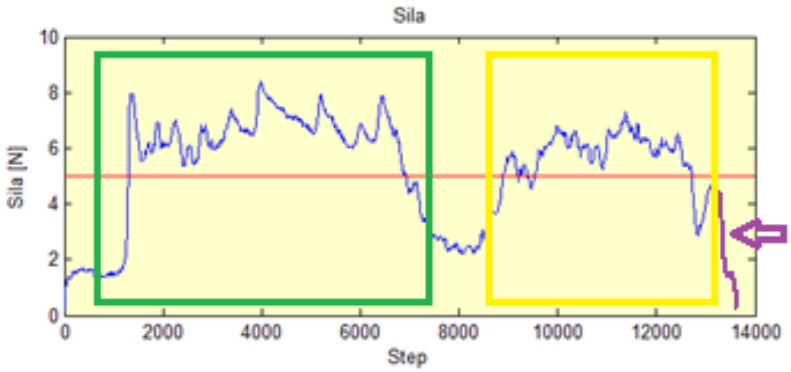
Slika 68. Kp=12 Ki=0. 1 Kd=1

Ponavljanje procesa bušenja sa istim parametrima u više navrata, nikada nije dalo niti približno sličan odziv. Zašto? Struktura kosti je jako promjenjiva, te je u svakom dijelu spužvasta kost druge čvrstoće. Isto tako, u pojedinim dijelovima kosti ima naslaga krvi koje uzrokuju proklizavanje bušilice uslijed male sile pritiska. U slučaju svježe kosti, tokom bušenja spužvasta kost se taloži na svrdlu i smanjuje odlijev odstranjenih komadića kosti.

Zašto nije uspjela regulacija bušenja sa PID regulatorom? Utjecaj manjih promjena parametara na ponašanje sustava može se zadovoljavajuće dobro kompenzirati klasičnim PID regulatorom. Međutim, znatnije promjene parametara sustava uzrokuju velika odstupanja od zadanog ponašanja. Problem sa PID regulatorima je što su linearni i njihova uspješnost u regulaciji nelinearnih, multivarijabilnih ili slučajnih sustava je varijabilna

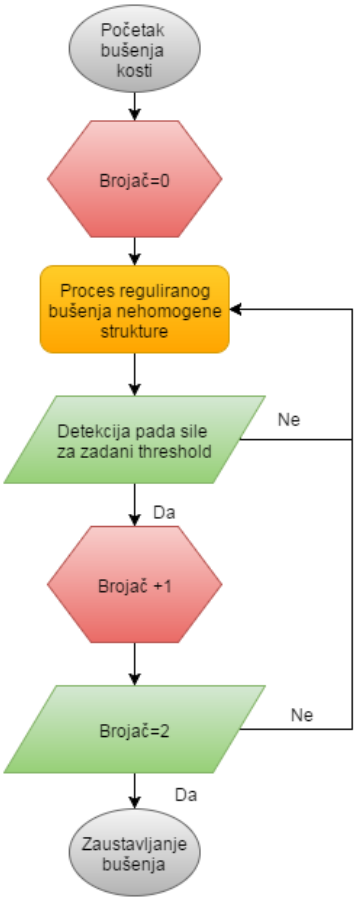
Pravovremeno zaustavljanje

Pravovremeno zaustavljanje bušilice prilikom bušenja nehomogene strukture ima više izazova. Ne postoji mogućnost jednostavnog prekida bušenja uslijed pada sile, jer pad sile ne znači proboj svrdla kroz kost, već nagli pad sile otpora bušenju, uzrokovan promjenom strukture nehomogenog materijala. Kao što je već navedeno, plosnata kost se sastoji od 3 dijela, mekanog međusloja spužvaste kosti između dvije puno tvrđe kompaktne kosti.



Slika 69. Promjena sile tokom bušenja plosnate kosti

Kao što je vidljivo na lici, odziv sile tokom bušenja ima velike oscilacije, te je potrebno razaznati zone bušenja pojedinih dijelova plosnate kosti, kako bi se mogao napisati algoritam bušenja koji će pravovremeno zaustaviti bušilicu i na taj način spriječiti moguće oštećenje mozga. Tri zone plosnate kosti je lako očitati sa Slike 69. Prva, zelena zona, predstavlja bušenje prve kompaktne kosti, gdje je potrebna veća sila kako bi se savladao otpor. Odma po proboju sila naglo pada zbog manjeg otpora. To je područje spužvaste kosti, nakon koje slijedi žuta zona, odnosno zona druge kompaktne kosti. Pravovremeno zaustavljanje bušenja je odmah po padu sile (označeno ljubičasto). Najvažniji termin kod ovog specifičnog problema jest threshold, odnosno vrijednost promjene sile. Ukoliko se predpostavi mala promjena sile, tj. prestanak bušenja nakon malog pada sile, spriječava se prodor svrdla unutar lubanje, ali postoji velika mogućnost prijevremenog detektiranja pad sile i završetka bušenja puno prije nego je ono uistinu završeno. Nasuprot tome, izbor velikog thresholda može za posljedicu imati imunost na male promjene sile, na prijevremeno zaustavljanje, ali tako prijeti opasnost od prekasnog zaustavljanja i velike oštete tkiva. Iskustvenom metodom, te ponavljanje bušenja na modelima može rezultirati najboljim izborom tresholda.



Slika 70. Dijagram toka za detekciju proboja plosnate kosti

8. ZAKLJUČAK

Zahtjevnost regulacije bušenja nehomogene strukture varira proporcionalno sa razlikom između mehaničkih svojstava slojeva predmeta koji se buši. Kod bušenja strukture poput pleksiglas/drvo/pleksiglas, u slučaju dobro podešenih parametara pojačanja, linearni PID regulator s lakoćom održava proces bušenja konstantnim. U slučaju ljudske kosti nalazimo jako veliku razliku u međuslojnim svojstvima i velike poremećaje, poput tekućine koja uzrokuje proklizavanje svrdla i promjene poroznosti same kosti, koji čine sustav nelinearnim. U tom slučaju primjena klasičnog linearnog PID regulatora nije najoptimalniji izbor.

U slučaju detekcije proboja svrdla kroz kost koristimo veliku razliku između svojstava slojeva kosti kako bi predpostavili trenutni položaj svrdla, a nagli pad sile kao signal proboja. Promjenom parametara bušenja, poput brzine i primjenjene sile bušenja povećavamo pad sile, odnosno razliku među slojevima i na taj način olakšavamo primjenu algoritma za pravovremeno zaustavljanje.

Također, korištenje velikog broja C++ skripti istovremeno pokazalo se nestabilnim izborom, jer zahtjevnost obrade podataka svake od njih ima za posljedicu prigušenje druge. Upravo zato dolazi do odstupanja i propada bušilice prilikom bušenja. Taj problem je uspješno riješen optimiziranjem koda. U konačnici, dobrom komunikacijom i optimalnim kodom omugućen je pravovremen i brz odziv sustava.

Kako bi se postigla najoptimalnija regulaciju bušenja kosti i održavanja sile bušenja konstantnom, potrebno je primjeniti neki od algoritama adaptivnog upravljanja. Adaptivno upravljanje je poseban oblik upravljanja u zatvorenoj petlji, gdje se informacije o upravljanom sustavu dobivaju tijekom radnog ciklusa, dok adaptivni regulator svojim djelovanjem kompenzira djelovanje poremećaja, parametara ili neke druge poremećajne veličine. Kompenzacija promjene parametara sustava ili nelinearnosti mogu se ostvariti pomoću regulatora s promjenjivim pojačanjem, samopodesivim regulatorom ili adaptivnim upravljanjem s referentnim modelom. Kao nastavak istraživanja predlaže se razvijanje i implementacija adaptivnog regulatora sa referentnim modelom koji bi bez problema riješio problem regulacije, upravo zbog lakoće ponavljanja procesa bušenja na modelima.

LITERATURA

[1] Kraut, B.: Strojarski priručnik, Tehnička knjiga Zagreb, 1970.

[2] A. Manolova: Description anatomique du mouvement

[3] PMF.: TCP/IP : http://www.phy.pmf.unizg.hr/~dandroic/nastava/ramr/poglavlje_3.pdf

[4] Iana.org: Service Name and Transport Protocol Port Number Registry

[5] F. R. Ong. K. Bouazza-Marouf / Mehatronics 9 (1999) 565-588

[6] Yeh-Liang Hsu / A modular Mechatronic System for automatic bone drilling

[7] dr. sc. Josip Kasač: skripta Vođenje tehničkih sustava

[8] dr. sc. Danijel Pavković: skripta Kalmanovi filteri / Upravljanje i regulacija

[9] Vladimir J. Šimunović: neurokirurgija

[10] Josip Paladino: Kompendij neurokirurgije

[11] Werner Platzer: Priručni anatomski atlas

[12] mag. ing. Filip Šuligoj: Podloge za vježbe iz Robotike

[13] mag. ing. Dominik Mihalinec: Relativno vođenje robota koristeći 3D vizijski sustav

[14] F.R. Ong, K. Bouazza-Marouf: The detection of drill bit break-through for the enhancement of safety in mechatronic assisted orthopaedic drilling

[15] FER: Podloge za vježbe iz Adaptivnog upravljanja

PRILOZI

I. CD-R disc

II. Nacrt pločice za prihvat senzora

III. Kod

PRILOZI

C++ KOD ZA KALIBRIRANJE TCP-a I GIBANJE PO OSI ALATA

glavni kod koji pokreće sve skripte istovremeno

```
#include <stdio.h>
#include <windows.h>
#include <process.h>          // needed for _beginthread()
#include <iostream>
#include <cstdlib>
#include <math.h>
#include <limits.h>

using namespace std;
void send(void *P);
void receive(void *P);
void stop(void *P);
void senzor(void *P);
int brojac = 0;
double faza;

int main()
{

	HANDLE hThreads[4];
	hThreads[0] = (HANDLE)_beginthread(send, 0, (void*)0);
	hThreads[1] = (HANDLE)_beginthread(receive, 0, (void*)0);
	hThreads[2] = (HANDLE)_beginthread(stop, 0, (void*)0);
	hThreads[3] = (HANDLE)_beginthread(senzor, 0, (void*)0);
	WaitForMultipleObjects(4, hThreads, TRUE, INFINITE);

	Sleep(2000);
}
```

očitanje položaja robotske ruke

```
#define WIN32_LEAN_AND_MEAN

#include <WinSock2.h>
#include <WS2tcpip.h>
#include <iostream>
#include <string>
#include <limits>
#include <algorithm>
#include <bitset>
#include <fstream>
#include <Eigen/Dense>
#include <Eigen/Geometry>
#include <vector>
#include <cmath>
#include <math.h>

// link with Ws2_32.lib
#pragma comment(lib, "Ws2_32.lib")

#define DEFAULT_PORT "30003"
#define DEFAULT_BUFFER_LENGTH     812


using namespace std;

// Global variable declaration:
int kx = 0, ky = 0, kz = 0;
```

```
double dx = 0, dy = 0, dz = 0;
double xx = 0, yy = 0, zz = 0;
double rx = 0, ry = 0, rz = 0;
double force = 0;
double vec_ang[6] = { 0, 0, 0, 0.4328, 0.3321, 1.2789 };
double vec_pos[6] = { 0, 0, 0, 0, 0, 0 };
int stp = 0;
double threshold = 15;
extern int input;
extern int tcp;
double theta;
int jj=0, dd=0, tt=0, tt2=0;



//-----------------------------------------------
		using Eigen::AngleAxisd;
		using Eigen::Matrix3d;
		using Eigen::Vector3d;
		using Eigen::MatrixXd;

		Vector3d RxRyRz;
		Vector3d T_1, T_2, T_3;
		Vector3d T_z1, T_z2;

		Matrix3d R_1;
		Matrix3d R_2;
		Matrix3d R_3;
		Matrix3d R_z1;
		Matrix3d R_z2;
		MatrixXd A(6,3);
		MatrixXd B(6,1);
		MatrixXd Xt1;
		MatrixXd Xt;
//-----------------------------------------------

double bitstring_to_double(const std::string& s)
{
	unsigned long long x = 0;
	for (std::string::const_iterator it = s.begin(); it != s.end(); ++it)
	{
		x = (x << 1) + (*it - '0');
	}
	double d;
	memcpy(&d, &x, 8);
	return d;
}

double Convert(char* recvbuf, int r) {

	string str;
	double d = 0;

	for (int i = r; i < (r + 8); i++)
	{
		unsigned long long x = recvbuf[i];
		unsigned long long a = 0;

		if (x >= 0)
			a = x;
		else
			a = x + 256;
```

```
			std::bitset<8> bin_x(a);

			string bitstring = bin_x.to_string<char, std::string::traits_type,
std::string::allocator_type>();
			str.append(bitstring);
		}
		d = bitstring_to_double(str);
		return d;
};

double get_target_tcp_pose(char* recvbuf, int r)
{
		string str;
		double d = 0;

		for (int i = r; i < (r + 8); i++)
		{
			unsigned long long x = recvbuf[i];
			unsigned long long a = 0;

			if (x >= 0)
				a = x;
			else
				a = x + 256;

			std::bitset<8> bin_x(a);

			string bitstring = bin_x.to_string<char, std::string::traits_type,
std::string::allocator_type>();
			str.append(bitstring);
		}
		d = bitstring_to_double(str);
		return d;
}

void tcp_odredivanje()
{

}
void matrica_tcp()
{

			if (input == 1 && jj == 0)
			{
				T_1 << xx, yy, zz;
				RxRyRz << rx, ry, rz;
				theta = sqrt(pow(rx, 2) + pow(ry, 2) + pow(rz, 2));
				R_1 = AngleAxisd(theta, RxRyRz.normalized()); //AxisAngle to DCM
				cout << "R_1" << endl;
				cout << R_1 << endl;
				cout << "T_1" << endl;
				cout << T_1 << endl;
				jj=1;
			}
			if (input == 2 && dd == 0)
			{
				T_2 << xx, yy, zz;
				RxRyRz << rx, ry, rz;
				theta = sqrt(pow(rx, 2) + pow(ry, 2) + pow(rz, 2));
				R_2 = AngleAxisd(theta, RxRyRz.normalized()); //AxisAngle to DCM
				cout << "R_2" << endl;
```

```
                        cout << R_2 << endl;
                        cout << "T_2" << endl;
                        cout << T_2 << endl;
                        dd=1;
                }
                if (input == 3 && tt == 0)
                {
                        T_3 << xx, yy, zz;
                        RxRyRz << rx, ry, rz;
                        theta = sqrt(pow(rx, 2) + pow(ry, 2) + pow(rz, 2));
                        R_3 = AngleAxisd(theta, RxRyRz.normalized()); //AxisAngle to DCM
                        cout << "R_3" << endl;
                        cout << R_3 << endl;
                        cout << "T_3" << endl;
                        cout << T_3 << endl;
                        tt=1;
                }
              if (input == 4 && tt2 == 0 && tt == 1 && dd == 1 && jj == 1)
                {

                                R_z1 = (R_2)-(R_1);
                                R_z2 = (R_3)-(R_2);
                                A << R_z1(0,0), R_z1(0,1), R_z1(0,2), R_z1(1,0), R_z1(1,1),
R_z1(1,2), R_z1(2,0), R_z1(2,1), R_z1(2,2),
                                        R_z2(0,0), R_z2(0,1), R_z2(0,2), R_z2(1,0),
R_z2(1,1), R_z2(1,2), R_z2(2,0), R_z2(2,1), R_z2(2,2);
                                T_z1 = (T_1)-(T_2);
                                T_z2 = (T_2)-(T_3);
                                B << T_z1(0,0), T_z1(1,0), T_z2(0,0), T_z2(1,0),
T_z2(2,0);
                                Xt1 = A.transpose()*A;
                                Xt = Xt1.inverse()*A.transpose()*B;

                                cout << Xt(0,0) * 100 << " " << Xt(1,0) * 100 << " " <<
Xt(2,0) * 100 << " " << endl;

                         tt2=1;
                  }
}

class Client {
public:
        Client(char* servername)
        {
                szServerName = "192.168.0.10";
                ConnectSocket = INVALID_SOCKET;
        }

        bool Start() {
                WSADATA wsaData;

                // Initialize Winsock
                int iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);
                if (iResult != 0)
                {
                        printf("WSAStartup failed: %d\n", iResult);
                        return false;
                }

                struct addrinfo      *result = NULL,
                        *ptr = NULL,
                        hints;
```

```
		ZeroMemory(&hints, sizeof(hints));
		hints.ai_family = AF_UNSPEC;
		hints.ai_socktype = SOCK_STREAM;
		hints.ai_protocol = IPPROTO_TCP;

		// Resolve the server address and port
		iResult = getaddrinfo(szServerName, DEFAULT_PORT, &hints, &result);
		if (iResult != 0)
		{
			printf("getaddrinfo failed: %d\n", iResult);
			WSACleanup();
			return false;
		}

		ptr = result;

		// Create a SOCKET for connecting to server
		ConnectSocket = socket(ptr->ai_family, ptr->ai_socktype, ptr-
>ai_protocol);

		if (ConnectSocket == INVALID_SOCKET)
		{
			printf("Error at socket(): %d\n", WSAGetLastError());
			freeaddrinfo(result);
			WSACleanup();
			return false;
		}

		// Connect to server
		iResult = connect(ConnectSocket, ptr->ai_addr, (int)ptr->ai_addrlen);

		if (iResult == SOCKET_ERROR)
		{
			closesocket(ConnectSocket);
			ConnectSocket = INVALID_SOCKET;
		}

		freeaddrinfo(result);

		if (ConnectSocket == INVALID_SOCKET)
		{
			printf("Unable to connect to server!\n");
			WSACleanup();
			return false;
		}

		return true;
	};

	// Free the resouces
	void Stop() {
		int iResult = shutdown(ConnectSocket, SD_SEND);

		if (iResult == SOCKET_ERROR)
		{
			printf("shutdown failed: %d\n", WSAGetLastError());
		}

		closesocket(ConnectSocket);
		WSACleanup();
	};
```

```
// Send message to server
bool Send(char* szMsg)
{

	int iResult = send(ConnectSocket, szMsg, strlen(szMsg), 0);

	if (iResult == SOCKET_ERROR)
	{
		printf("send failed: %d\n", WSAGetLastError());
		Stop();
		return false;
	}

	return true;
};

// Receive message from server
bool Recv()
{

	char recvbuf[DEFAULT_BUFFER_LENGTH];
	int iResult = recv(ConnectSocket, recvbuf, DEFAULT_BUFFER_LENGTH, 0);

	int x = 540, y = 548, z = 556;
	int a = 612, b = 620, c = 628;
	int e = 588, f = 596, g = 604;


	if (iResult > 0)
	{

		dx = Convert(recvbuf, x);

		dy = Convert(recvbuf, y);

		dz = Convert(recvbuf, z);

		rx = Convert(recvbuf, a);

		ry = Convert(recvbuf, b);

		rz = Convert(recvbuf, c);

		xx = Convert(recvbuf, e);

		yy = Convert(recvbuf, f);

		zz = Convert(recvbuf, g);

		matrica_tcp();

		//cout << "print " << xx << " " << yy << " " << zz << endl;
		//cout << "print " << rx << " " << ry << " " << rz << endl;
		//cout << dx << " " << dy << " " << dz << '\n';
		//cout << "Force: " << force << endl;

	}

	return true;
}
```

```
private:
	char* szServerName;
	SOCKET ConnectSocket;
};

// KOD
void receive(void *P)
{
	string msg, msg1, msg2;
	int i = 0;
	Client client("192.168.0.100");
	string recived;


	if (!client.Start())
		return;

	while (true)
	{
		client.Recv();
		if (input == 6)
			jj=0, dd=0, tt=0, tt2=0;
	}

	client.Stop();

	getchar();
	return;
}
```

očitanje vrijednosti sile sa senzora

```
#define WIN32_LEAN_AND_MEAN

#include <WinSock2.h>
#include <WS2tcpip.h>
#include <iostream>
#include <string>
#include <limits>
#include <algorithm>
#include <bitset>
#include <fstream>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <sstream>

// link with Ws2_32.lib
#pragma comment(lib, "Ws2_32.lib")

#define DEFAULT_PORT "63351"
#define DEFAULT_BUFFER_LENGTH     512


using namespace std;

// Global variable declaration:

double sensor = 0;


class Client3 {
```

```
public:
	Client3(char* servername)
	{
		szServerName = "192.168.0.10";
		ConnectSocket = INVALID_SOCKET;
	}

	bool Start() {
		WSADATA wsaData;

		// Initialize Winsock
		int iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);
		if (iResult != 0)
		{
			printf("WSAStartup failed: %d\n", iResult);
			return false;
		}

		struct addrinfo	*result = NULL,
			*ptr = NULL,
			hints;

		ZeroMemory(&hints, sizeof(hints));
		hints.ai_family = AF_UNSPEC;
		hints.ai_socktype = SOCK_STREAM;
		hints.ai_protocol = IPPROTO_TCP;

		// Resolve the server address and port
		iResult = getaddrinfo(szServerName, DEFAULT_PORT, &hints, &result);
		if (iResult != 0)
		{
			printf("getaddrinfo failed: %d\n", iResult);
			WSACleanup();
			return false;
		}

		ptr = result;

		// Create a SOCKET for connecting to server
		ConnectSocket = socket(ptr->ai_family, ptr->ai_socktype, ptr-
>ai_protocol);

		if (ConnectSocket == INVALID_SOCKET)
		{
			printf("Error at socket(): %d\n", WSAGetLastError());
			freeaddrinfo(result);
			WSACleanup();
			return false;
		}

		// Connect to server
		iResult = connect(ConnectSocket, ptr->ai_addr, (int)ptr->ai_addrlen);

		if (iResult == SOCKET_ERROR)
		{
			closesocket(ConnectSocket);
			ConnectSocket = INVALID_SOCKET;
		}

		freeaddrinfo(result);

		if (ConnectSocket == INVALID_SOCKET)
```

```
		{
			printf("Unable to connect to server!\n");
			WSACleanup();
			return false;
		}

		return true;
	};

	// Free the resouces
	void Stop() {
		int iResult = shutdown(ConnectSocket, SD_SEND);

		if (iResult == SOCKET_ERROR)
		{
			printf("shutdown failed: %d\n", WSAGetLastError());
		}

		closesocket(ConnectSocket);
		WSACleanup();
	};


	// Send message to server
	bool Send(char* szMsg)
	{

		int iResult = send(ConnectSocket, szMsg, strlen(szMsg), 0);

		if (iResult == SOCKET_ERROR)
		{
			printf("send failed: %d\n", WSAGetLastError());
			Stop();
			return false;
		}

		return true;
	};

	// Receive message from server
	bool Recv()
	{
		char recvbuf[DEFAULT_BUFFER_LENGTH];
		int iResult = recv(ConnectSocket, recvbuf, DEFAULT_BUFFER_LENGTH, 0);

		if (iResult > 0)
		{

			char b1[10];
			char b2[20];
			char b3[20];

			stringstream ss, ss1, ss2;
			string s, s1, s2;
			//cout << recvbuf << endl;

			int i, n, n1=0;
			int i2;
			int i3=0;

			for (i=0; i<512; i++)
				{
```

```
                                        if (recvbuf[i]=='(')  // && i3==0
                {
                                                for (n=55;n<70;n++)
                                                        {if(recvbuf[i+n]==')')
                                                                n1=1;
                                                        }
                                                if(n1==1)
                                                {
                                                i3=1;
                                                for(i2=1; i2<10; i2++)
                                                        b1[i2-1]=recvbuf[i+i2];
                                                        //cout << b1 << endl;

                                                for(i2=12; i2<21; i2++)
                                                        b2[i2-12]=recvbuf[i+i2];
                                                        //cout << b2 << endl;

                                                for(i2=24; i2<33; i2++)
                                                        b3[i2-24]=recvbuf[i+i2];
                                                        //cout << b3 << endl;

                                                //cout << "---------" << endl;
                                                //cout << recvbuf << endl;
                                                //cout << "---------" << endl;
                                                ss << b1;
                                                ss >> s;
                                                ss1 << b2;
                                                ss1 >> s1;
                                                ss2 << b3;
                                                ss2 >> s2;

                                                double n1;
                                                double n2;
                                                double n3;

                                                n1 = atof(b1);
                                                n2 = atof(b2);
                                                n3 = atof(b3);
                                                if (n1!=0 && n2!=0 && n3!=0)
                                                        {       double sensor =
                                                sqrt(pow(n1,2) + pow(n2,2) + pow(n3,2));
                                                        //Sleep(6000);

                                                        ofstream myfile;

                                        myfile.open ("force.txt", ios::app);
                                                        myfile << sensor;
                                                        myfile << ",";
                                                        myfile.close();
                                                        }
                                                }
                                        }

                                }
                                  i3=0; n1=0;
                }

                return true;
        }

private:
        char* szServerName;
```

```
	SOCKET ConnectSocket;
};

// KOD
void senzor(void *P)
{

	Client3 Client3("192.168.0.100");
	string recived;


	if (!Client3.Start())
		return;

	while (true)
	{
				Client3.Recv();
	}


	Client3.Stop();

	getchar();
	return;
}
```

gibanje robota i izbornik

```
#define WIN32_LEAN_AND_MEAN

#include <WinSock2.h>
#include <WS2tcpip.h>
#include <iostream>
#include <string>
#include <limits>
#include <algorithm>
#include <bitset>

// link with Ws2_32.lib
#pragma comment(lib, "Ws2_32.lib")

#define DEFAULT_PORT "30003"
#define DEFAULT_BUFFER_LENGTH	812

//extern double force;
using namespace std;


// Global variable declaration:
extern double xx, yy, zz;
extern double rx, ry, rz;
extern double force;
int k=0;
int izbor, izbor2, izbor3;
int input=0;
int tcp=0;
int g_z=0;
extern int gg;
extern string msg11;
double korak=0.02;
double a1, b1, c1;
void showWelcome()
{
```

```
	cout << "Program je aktivan." << endl;
}

void showMenu()
{
                    cout << "Molim odaberite funkciju:" << endl
                     << "1. Kalibracija TCP-a" << endl
                     << "2. Gibanje" << endl
                     << "3. Home" << endl
                     << "4. Izlaz" << endl;
}

void Gibanje()
{
                    cout << "Molim odaberite funkciju:" << endl
                     << "1. Korak (cm)" << endl
                     << "2. +Z" << endl
                     << "3. -Z" << endl
                     << "4. +X" << endl
                     << "5. -X" << endl
                     << "6. +Y" << endl
                     << "7. -Y" << endl
                     << "8. Izlaz" << endl;
}

void Home()
{
                    cout << "Molim odaberite funkciju:" << endl
                     << "1. Home_far" << endl
                     << "2. Home_close" << endl
                     << "3. Izlaz" << endl;
}

void PressEnterToContinue()
{
	int c;
	//cout << "Robot je u HOME položaju" << endl;
	//cout << "Molim pritisnite ENTER za nastavak:" << endl;
	fflush(stdout);
	do c = getchar(); while ((c != '\n') && (c != EOF));
}

void PressEnterToContinue4()
{
	int c;
	//cout << "Robot je u HOME položaju" << endl;
	//cout << "Molim pritisnite ENTER za nastavak:" << endl;
	fflush(stdout);
	do c = getchar(); while ((c != '\n') && (c != EOF));
}

void PressEnterToContinue0()
{

	int c;
	//cout << "Robot je u HOME položaju" << endl;
	cout << "Molim postavite robota u PRVU poziciju i pritisnite enter." << endl;
	fflush(stdout);
	do c = getchar(); while ((c != '\n') && (c != EOF));
}

void PressEnterToContinue2()
```

```
{
	int c;
	//cout << "Robot je u HOME položaju" << endl;
	cout << "Molim postavite robota u DRUGU poziciju i pritisnite enter." << endl;
	//cin.ignore(std::numeric_limits<std::streamsize>::max(),'\n');
	fflush(stdout);
	do c = getchar(); while ((c != '\n') && (c != EOF));
}

void PressEnterToContinue3()
{
	int c;
	//cout << "Robot je u HOME položaju" << endl;
	cout << "Molim postavite robota u TRECU poziciju i pritisnite enter." << endl;
	fflush(stdout);
	do c = getchar(); while ((c != '\n') && (c != EOF));
}

void PressEnterToContinue1()
{
	int c;
	//cout << "Robot je u HOME položaju" << endl;
	fflush(stdout);
	do c = getchar(); while ((c != '\n') && (c != EOF));
}

void tcp_kalibracija()
{
			PressEnterToContinue4();
			PressEnterToContinue0();
				input=1;
				Sleep(500);
			PressEnterToContinue2();
				input=2;
				Sleep(500);
			PressEnterToContinue3();
				input=3;
				Sleep(500);
			cout << "Kalibracija je zavrsena." << endl;
				input=4;
				Sleep(500);
			PressEnterToContinue4();
}

class Client1 {
public:
	Client1(char* servername)
	{
		szServerName = "192.168.0.10";
		ConnectSocket = INVALID_SOCKET;
	}

	bool Start() {
		WSADATA wsaData;

		// Initialize Winsock
		int iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);
		if (iResult != 0)
		{
			printf("WSAStartup failed: %d\n", iResult);
			return false;
		}
```

```
		struct addrinfo     *result = NULL,
			*ptr = NULL,
			hints;

		ZeroMemory(&hints, sizeof(hints));
		hints.ai_family = AF_UNSPEC;
		hints.ai_socktype = SOCK_STREAM;
		hints.ai_protocol = IPPROTO_TCP;

		// Resolve the server address and port
		iResult = getaddrinfo(szServerName, DEFAULT_PORT, &hints, &result);
		if (iResult != 0)
		{
			printf("getaddrinfo failed: %d\n", iResult);
			WSACleanup();
			return false;
		}

		ptr = result;

		// Create a SOCKET for connecting to server
		ConnectSocket = socket(ptr->ai_family, ptr->ai_socktype, ptr-
>ai_protocol);

		if (ConnectSocket == INVALID_SOCKET)
		{
			printf("Error at socket(): %d\n", WSAGetLastError());
			freeaddrinfo(result);
			WSACleanup();
			return false;
		}

		// Connect to server
		iResult = connect(ConnectSocket, ptr->ai_addr, (int)ptr->ai_addrlen);

		if (iResult == SOCKET_ERROR)
		{
			closesocket(ConnectSocket);
			ConnectSocket = INVALID_SOCKET;
		}

		freeaddrinfo(result);

		if (ConnectSocket == INVALID_SOCKET)
		{
			printf("Unable to connect to server!\n");
			WSACleanup();
			return false;
		}

		return true;
	};

	// Free the resouces
	void Stop() {
		int iResult = shutdown(ConnectSocket, SD_SEND);

		if (iResult == SOCKET_ERROR)
		{
			printf("shutdown failed: %d\n", WSAGetLastError());
		}
```

```
		closesocket(ConnectSocket);
		WSACleanup();
	};


	// Send message to server
	bool Send(char* szMsg)
	{

		int iResult = send(ConnectSocket, szMsg, strlen(szMsg), 0);

		if (iResult == SOCKET_ERROR)
		{
			printf("send failed: %d\n", WSAGetLastError());
			Stop();
			return false;
		}

		return true;
	};



private:
	char* szServerName;
	SOCKET ConnectSocket;
};

// KOD
void send(void *P)
{

	Client1 Client1("192.168.0.100");
	string recived;

	if (!Client1.Start())
		return;

	while (true)
	{
		showWelcome();
		mylabel1:
		showMenu();
		cin >> izbor;
		while (izbor < 1 || izbor >4)
			{ cout << "Nepoznati unos, molim ponovite." << endl;
			  goto mylabel1;
				}
		if (izbor==4)
			goto mylabel1;
		if (izbor==1)
		{
			tcp_kalibracija();
		}
		if (izbor==2)
		{
				mylabel2:
				Gibanje();
				cin >> izbor2;
				while (izbor2 < 1 || izbor2 >8)
				{ cout << "Nepoznati unos, molim ponovite." << endl;
```

```
  goto mylabel2;
}
if (izbor2==1)
	{
		cout << "Molim upisite korak u centimetrima:
(npr. 2)" << endl;
		double korak1;
		cin >> korak1;
		korak = korak1/100;
		cout << "Korak je: " << korak << "m." <<endl;
		goto mylabel2;
	}
if (izbor2==2)
	{
		cout << "Pritisnite ENTER za gibanje " <<
	endl;
		g_z=1;
		a1=-korak; b1=0; c1=0;
			if (gg = 2)
			{
			PressEnterToContinue1();
			PressEnterToContinue1();
			cout << msg11 << endl;
			Client1.Send((char*)msg11.c_str());
			Sleep(1000);

			cout << "Gibanje zavrseno, enter za
		nastavak" << endl;
			PressEnterToContinue1();
			gg=0;
			}
		g_z=0;
	}
if (izbor2==3)
	{
		cout << "Pritisnite ENTER za gibanje " <<
		endl;
		g_z=1;
		a1=korak; b1=0; c1=0;
			if (gg = 2)
			{
			PressEnterToContinue1();
			PressEnterToContinue1();
			cout << msg11 << endl;
			Client1.Send((char*)msg11.c_str());
			Sleep(1000);
                                                cout <<
		"Gibanje zavrseno, enter za nastavak" << endl;
			PressEnterToContinue1();
			gg=0;
			}
		g_z=0;
	}
if (izbor2==4)
	{
		cout << "Pritisnite ENTER za gibanje " <<
endl;
		g_z=1;
		a1=0; b1=-korak; c1=0;
			if (gg = 2)
			{
			PressEnterToContinue1();
```

```
                                        PressEnterToContinue1();
                                        cout << msg11 << endl;
                                        Client1.Send((char*)msg11.c_str());
                                        Sleep(1000);

                                        cout << "Gibanje zavrseno, enter za
                                nastavak" << endl;
                                        PressEnterToContinue1();
                                        gg=0;
                                        }
                                g_z=0;
                        }
                if (izbor2==5)
                        {
                                cout << "Pritisnite ENTER za gibanje " <<
                                endl;
                                g_z=1;
                                a1=0; b1=korak; c1=0;
                                        if (gg = 2)
                                        {
                                        PressEnterToContinue1();
                                        PressEnterToContinue1();
                                        cout << msg11 << endl;
                                        Client1.Send((char*)msg11.c_str());
                                        Sleep(1000);

                                        cout << "Gibanje zavrseno, enter za
                                nastavak" << endl;
                                        PressEnterToContinue1();
                                        gg=0;
                                        }
                                g_z=0;
                        }
                if (izbor2==6)
                        {
                                cout << "Pritisnite ENTER za gibanje " <<
                                endl;
                                g_z=1;
                                a1=0; b1=0; c1=-korak;
                                        if (gg = 2)
                                        {
                                        PressEnterToContinue1();
                                        PressEnterToContinue1();
                                        cout << msg11 << endl;
                                        Client1.Send((char*)msg11.c_str());
                                        Sleep(1000);

                                        cout << "Gibanje zavrseno, enter za
                                nastavak" << endl;
                                        PressEnterToContinue1();
                                        gg=0;
                                        }
                                g_z=0;
                        }
                if (izbor2==7)
                        {
                                cout << "Pritisnite ENTER za gibanje " <<
                                endl;
                                g_z=1;
                                a1=0; b1=0; c1=korak;
                                        if (gg = 2)
                                        {
```

```
                                        PressEnterToContinue1();
                                        PressEnterToContinue1();
                                        cout << msg11 << endl;
                                        Client1.Send((char*)msg11.c_str());
                                        Sleep(1000);

                                        cout << "Gibanje zavrseno, enter za
                                  nastavak" << endl;
                                        PressEnterToContinue1();
                                        gg=0;
                                        }
                                  g_z=0;
                            }
                      if (izbor2==8)
                            goto mylabel1;

      }
      if (izbor==3)
      {
            string msgdalje, msgblize;
                                  msgblize = "movel([-5.535376069617674, -
            2.372912901639345, 2.4692891597444566, -2.6496088097006467, -
            1.0348388310400438, -3.29857514701704], a=0.08, v=0.004)\n";
            msgdalje = "movel([-5.535376069736867, -2.3715609923873813,
            2.26499583420581, -2.4466673933746725, -1.034838830940896, -
            3.2985751470940103], a=0.08, v=0.009)\n";
            mylabel3:
            Home();
                  cin >> izbor3;
                  while (izbor3 < 1 || izbor3 >3)
                        { cout << "Nepoznati unos, molim ponovite." << endl;
                        goto mylabel3;}
                  if (izbor3==3)
                        goto mylabel1;
                  if (izbor3==1)
                  {
                        cout << "Odlazak u Home poziciju." << endl;
                        PressEnterToContinue1();
                        PressEnterToContinue1();
                        Client1.Send((char*)msgdalje.c_str());
                        Sleep(3000);
                                                cout << "Gibanje zavrseno, enter
            za nastavak" << endl;
                        PressEnterToContinue1();
                  }
                  if (izbor3==2)
                  {
                        cout << "Odlazak u Home poziciju." << endl;
                        PressEnterToContinue1();
                        PressEnterToContinue1();
                        Client1.Send((char*)msgblize.c_str());
                        Sleep(3000);
                                        cout << "Gibanje zavrseno, enter za
      nastavak" << endl;
                        PressEnterToContinue1();
                  }
      }
      //brzine za busenje   a=0.019, v=0.0009
      input=6;
      //cout << string(100,'\n');  //ovo tu cisti ekran
      goto mylabel1;
```

```
                //Client1.Send((char*)msg1.c_str());

        }

        Client1.Stop();

        getchar();
        return;
}
```

transformacija gibanja

```
#define WIN32_LEAN_AND_MEAN

#include <WinSock2.h>
#include <WS2tcpip.h>
#include <iostream>
#include <string>
#include <limits>
#include <algorithm>
#include <bitset>
#include <Eigen/Dense>
#define WIN32_LEAN_AND_MEAN
#include <bitset>
#include <Eigen/Dense>
#include <Eigen/Geometry>
#include <vector>
#include <cmath>
#include <math.h>

// link with Ws2_32.lib
#pragma comment(lib, "Ws2_32.lib")

#define DEFAULT_PORT "30003"
#define DEFAULT_BUFFER_LENGTH     812


using namespace std;

// Global variable declaration:
int kxn = 0, kyn = 0, kzn = 0;
double dxn = 0, dyn = 0, dzn = 0;
double xxn = 0, yyn = 0, zzn = 0;
double rxn = 0, ryn = 0, rzn = 0;
volatile double forcen = 0;
double theta_1n, theta_2n, theta_3n=0;
double a = 0, b = 0, c = 0, u = 0, v = 0, w = 0;
double vec_angn[6] = { 0, 0, 0, 0.4328, 0.3321, 1.2789 };
double vec_posn[6] = { 0, 0, 0, 0, 0, 0 };
double vec_moven[6] = { a, b, c, u, v, w };
double polja_slanjen[6];
double polja_slanjen1n[6];
int gg=1;
string msg11;
string msg2;
extern int g_z;
extern double a1, b1, c1;



//-------------------------------------------------------------
        using Eigen::AngleAxisd;
```

```
	using Eigen::Matrix3d;
	using Eigen::Matrix4d;
	using Eigen::Vector3d;
	using Eigen::VectorXd;

	Vector3d rxnrynrzn1;
	Vector3d rxnrynrzn2;
	Vector3d rxnrynrzn3;

	Matrix3d Robot_DCM_1n;
	Matrix3d Robot_DCM_2n;
	Matrix3d Robot_DCM_3n;
	Matrix3d P1n_3x3;
	Matrix3d P2n_3x3;
	Matrix3d P3n_3x3;
	Matrix4d nT1;
	Matrix4d nT2;
	Matrix4d nT12;
	Matrix4d nT13;
	Matrix4d nT14;
	Matrix4d nT15;
	Matrix4d nT16;

//-------------------------------------------------------------



double bitstring_to_doublen(const std::string& s)
{
	unsigned long long x = 0;
	for (std::string::const_iterator it = s.begin(); it != s.end(); ++it)
	{
		x = (x << 1) + (*it - '0');
	}
	double d;
	memcpy(&d, &x, 8);
	return d;
}

double Convertn(char* recvbuf, int r) {

	string str;
	double d = 0;

	for (int i = r; i < (r + 8); i++)
	{
		unsigned long long x = recvbuf[i];
		unsigned long long a = 0;

		if (x >= 0)
			a = x;
		else
			a = x + 256;

		std::bitset<8> bin_x(a);

		string bitstring = bin_x.to_string<char, std::string::traits_type,
std::string::allocator_type>();
		str.append(bitstring);
	}
	d = bitstring_to_doublen(str);
```

```
	return d;
};


void PressEnterToContinue123()
{
	int c;
	//cout << "Robot je u HOME položaju" << endl;
	cout << "Molim pritisnite ENTER za nastavak:" << endl;
	fflush(stdout);
	do c = getchar(); while ((c != '\n') && (c != EOF));
}

void pose_trans()
{
	//-0.7011, -2.7890, -0.8481;
	//0.8551, -1.6113, -0.6113;
	//0.0915, -2.3597, -0.7873;
	rxnrynrzn1 << 1.3368, -1.8366, 0.1593;
	rxnrynrzn2 << rxn, ryn, rzn;
	rxnrynrzn3 << 0, 0, 0;
	//Vectorxnd pomak;


	double angle2n;


	theta_1n = sqrt(pow(1.3368, 2) + pow(-1.8366, 2) + pow(0.1593, 2));
	theta_2n = sqrt(pow(rxn, 2) + pow(ryn, 2) + pow(rzn, 2));

	Robot_DCM_1n = AngleAxisd(theta_1n, rxnrynrzn1.normalized()); //AxisAngle to DCM
	Robot_DCM_2n = AngleAxisd(theta_2n, rxnrynrzn2.normalized()); //AxisAngle to DCM

	nT1 << Robot_DCM_1n(0, 0), Robot_DCM_1n(0, 1), Robot_DCM_1n(0, 2), 0,
Robot_DCM_1n(1, 0), Robot_DCM_1n(1, 1), Robot_DCM_1n(1, 2), 0, Robot_DCM_1n(2, 0),
Robot_DCM_1n(2, 1), Robot_DCM_1n(2, 2), 0, 0, 0, 0, 1;
	//current x,y,z,
	nT2 << Robot_DCM_2n(0, 0), Robot_DCM_2n(0, 1), Robot_DCM_2n(0, 2), 0,
Robot_DCM_2n(1, 0), Robot_DCM_2n(1, 1), Robot_DCM_2n(1, 2), 0, Robot_DCM_2n(2, 0),
Robot_DCM_2n(2, 1), Robot_DCM_2n(2, 2), 0, 0, 0, 0, 1;
	//nT2 << Robot_DCM_2n(0, 0), Robot_DCM_2n(0, 1), Robot_DCM_2n(0, 2), xxn,
Robot_DCM_2n(1, 0), Robot_DCM_2n(1, 1), Robot_DCM_2n(1, 2), yyn, Robot_DCM_2n(2, 0),
Robot_DCM_2n(2, 1), Robot_DCM_2n(2, 2), zzn, 0, 0, 0, 1;
	//
	//cout << "Pravi TCP\n" << endl;
	//cout << xxn<< " " << yyn << " " << zzn << " " << rxn << " " << ryn << " " <<
rzn<< endl;



	//cout << "nT1\n" << endl;
	//cout << nT1 << endl;
	//cout << "nT2\n" << endl;
	//cout << nT2 << endl;

		nT12 = nT2*nT1;



		 polja_slanjen1n[0]=nT12(0,3);
		 polja_slanjen1n[1]=nT12(1,3);
		 polja_slanjen1n[2]=nT12(2,3);
```

```
                P2n_3x3=nT12.block<3,3>(0,0);
                angle2n=acos((P2n_3x3(0,0)+P2n_3x3(1,1)+P2n_3x3(2,2)-1)/2);
                polja_slanjen1n[3]=(P2n_3x3(2,1)-P2n_3x3(1,2))/sqrt( pow(P2n_3x3(2,1)-
P2n_3x3(1,2),2) + pow(P2n_3x3(0,2)-P2n_3x3(2,0),2) + pow(P2n_3x3(1,0)-P2n_3x3(0,1),2)
)*angle2n;
                polja_slanjen1n[4]=(P2n_3x3(0,2)-P2n_3x3(2,0))/sqrt( pow(P2n_3x3(2,1)-
P2n_3x3(1,2),2) + pow(P2n_3x3(0,2)-P2n_3x3(2,0),2) + pow(P2n_3x3(1,0)-P2n_3x3(0,1),2)
)*angle2n;
                polja_slanjen1n[5]=(P2n_3x3(1,0)-P2n_3x3(0,1))/sqrt( pow(P2n_3x3(2,1)-
P2n_3x3(1,2),2) + pow(P2n_3x3(0,2)-P2n_3x3(2,0),2) + pow(P2n_3x3(1,0)-P2n_3x3(0,1),2)
)*angle2n;

//------------------------------
        //      cout << polja_slanjen1n[0]<< " " << polja_slanjen1n[1] << " " <<
polja_slanjen1n[2] << " " << polja_slanjen1n[3] << " " << polja_slanjen1n[4] << " " <<
polja_slanjen1n[5]<< endl;


//      cout << "nT2*nT1\n" << endl;
//      cout << nT12 <<endl;

        a=a1;
        b=b1;
        c=c1;

        nT13 << 0, 0, 0, a, 0, 0, 0, b, 0, 0, 0, c, 0, 0, 0, 1;

        //cout << "nT13\n" << endl;
        //cout << nT13 << endl;

        nT14 = nT12 * nT13;

        /*cout << "nT14\n" << endl;
        cout << nT14 << endl;*/


        //nT14(0,0)=0; nT14(0,1)=0; nT14(0,2)=0;
        //nT14(1,0)=0; nT14(1,1)=0; nT14(1,2)=0;
        //nT14(2,0)=0; nT14(2,1)=0; nT14(2,2)=0;

        //cout << "nT14\n" << endl;
        //cout << nT14 << endl;


                polja_slanjen[0]=nT14(0,3);
                polja_slanjen[1]=nT14(1,3);
                polja_slanjen[2]=nT14(2,3);

                double xxnx = xxn+polja_slanjen[0];
                double yyny = yyn+polja_slanjen[1];
                double zznz = zzn+polja_slanjen[2];



                ostringstream msg1;
                msg1.str("");
                //string msg3;
                msg1 << "movel(p[" << xxnx << ", " << yyny << ", " << zznz << ", " <<
rxn << ", " << ryn << ", " << rzn << "], a=0.08, v=0.009)\n";
                msg11 = msg1.str();
```

```
                //msg3 = "movel([-0.0611261, -0.170528, 0.650519, 0.612863, -2.00207,
0.400816], a=0.08, v=0.009)\n";
                //cout << msg3 << endl;
                gg=2;
                g_z=0;

                //Sleep(90000);

         return;
}

void get_target_tcp_pose()
{

        double vec_posn[6] = { xxn, yyn, zzn, rxn, ryn, rzn };

        return;
}

class Client2 {
public:
        Client2(char* servername)
        {
                szServerName = "192.168.0.10";
                ConnectSocket = INVALID_SOCKET;
        }

        bool Start() {
                WSADATA wsaData;

                // Initialize Winsock
                int iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);
                if (iResult != 0)
                {
                        printf("WSAStartup failed: %d\n", iResult);
                        return false;
                }

                struct addrinfo      *result = NULL,
                        *ptr = NULL,
                        hints;

                ZeroMemory(&hints, sizeof(hints));
                hints.ai_family = AF_UNSPEC;
                hints.ai_socktype = SOCK_STREAM;
                hints.ai_protocol = IPPROTO_TCP;

                // Resolve the server address and port
                iResult = getaddrinfo(szServerName, DEFAULT_PORT, &hints, &result);
                if (iResult != 0)
                {
                        printf("getaddrinfo failed: %d\n", iResult);
                        WSACleanup();
                        return false;
                }

                ptr = result;

                // Create a SOCKET for connecting to server
                ConnectSocket = socket(ptr->ai_family, ptr->ai_socktype, ptr-
>ai_protocol);
```

```
		if (ConnectSocket == INVALID_SOCKET)
		{
			printf("Error at socket(): %d\n", WSAGetLastError());
			freeaddrinfo(result);
			WSACleanup();
			return false;
		}

		// Connect to server
		iResult = connect(ConnectSocket, ptr->ai_addr, (int)ptr->ai_addrlen);

		if (iResult == SOCKET_ERROR)
		{
			closesocket(ConnectSocket);
			ConnectSocket = INVALID_SOCKET;
		}

		freeaddrinfo(result);

		if (ConnectSocket == INVALID_SOCKET)
		{
			printf("Unable to connect to server!\n");
			WSACleanup();
			return false;
		}

		return true;
	};

	// Free the resouces
	void Stop() {
		int iResult = shutdown(ConnectSocket, SD_SEND);

		if (iResult == SOCKET_ERROR)
		{
			printf("shutdown failed: %d\n", WSAGetLastError());
		}

		closesocket(ConnectSocket);
		WSACleanup();
	};


	// Send message to server
	bool Send(char* szMsg)
	{

		int iResult = send(ConnectSocket, szMsg, strlen(szMsg), 0);

		if (iResult == SOCKET_ERROR)
		{
			printf("send failed: %d\n", WSAGetLastError());
			Stop();
			return false;
		}

		return true;
	};

	// Receive message from server
	bool Recv()
	{
```

```
		char recvbuf[DEFAULT_BUFFER_LENGTH];
		int iResult = recv(ConnectSocket, recvbuf, DEFAULT_BUFFER_LENGTH, 0);
		int x = 540, y = 548, z = 556;
		int a = 612, b = 620, c = 628;
		int e = 588, f = 596, g = 604;

		if (iResult > 0)
		{

			dxn = Convertn(recvbuf, x);
			dyn = Convertn(recvbuf, y);
			dzn = Convertn(recvbuf, z);

			forcen = sqrt(dxn*dxn + dyn*dyn + dzn*dzn);

			rxn = Convertn(recvbuf, a);
			ryn = Convertn(recvbuf, b);
			rzn = Convertn(recvbuf, c);
			xxn = Convertn(recvbuf, e);
			yyn = Convertn(recvbuf, f);
			zzn = Convertn(recvbuf, g);

			//cout << dxn << " " << dyn << " " << dzn << '\n';

			pose_trans();

		}

		return true;
	}

private:
	char* szServerName;
	SOCKET ConnectSocket;
};

// KOD
void stop(void *P)
{


	Client2 Client2("192.168.0.100");
	string recived;

	if (!Client2.Start())
		return;
	int g=0;
	while (true)
	{

		if(g_z==1)
			{
			Client2.Recv();
			}

	}


	Client2.Stop();

	getchar();
	return;
```

```
}
```

gibanje robota u slučaju regulacije bušenja

```
#define WIN32_LEAN_AND_MEAN

#include <WinSock2.h>
#include <WS2tcpip.h>
#include <iostream>
#include <string>
#include <limits>
#include <algorithm>
#include <bitset>

// link with Ws2_32.lib
#pragma comment(lib, "Ws2_32.lib")

#define DEFAULT_PORT "30001"
#define DEFAULT_BUFFER_LENGTH     812

//extern double force;
using namespace std;


// Global variable declaration:
extern double xx, yy, zz;
extern double rx, ry, rz;
extern double force;
int k=0;
int go=0;

void PressEnterToContinue()
{
	int c;
	//cout << "Robot je u HOME položaju" << endl;
	cout << "Molim pritisnite ENTER za nastavak:" << endl;
	fflush(stdout);
	do c = getchar(); while ((c != '\n') && (c != EOF));
}

class Client1 {
public:
	Client1(char* servername)
	{
		szServerName = "192.168.0.10";
		ConnectSocket = INVALID_SOCKET;
	}

	bool Start() {
		WSADATA wsaData;

		// Initialize Winsock
		int iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);
		if (iResult != 0)
		{
			printf("WSAStartup failed: %d\n", iResult);
			return false;
		}

		struct addrinfo     *result = NULL,
			*ptr = NULL,
			hints;
```

```
		ZeroMemory(&hints, sizeof(hints));
		hints.ai_family = AF_UNSPEC;
		hints.ai_socktype = SOCK_STREAM;
		hints.ai_protocol = IPPROTO_TCP;

		// Resolve the server address and port
		iResult = getaddrinfo(szServerName, DEFAULT_PORT, &hints, &result);
		if (iResult != 0)
		{
			printf("getaddrinfo failed: %d\n", iResult);
			WSACleanup();
			return false;
		}

		ptr = result;

		// Create a SOCKET for connecting to server
					ConnectSocket = socket(ptr->ai_family, ptr-
			>ai_socktype, ptr->ai_protocol);

		if (ConnectSocket == INVALID_SOCKET)
		{
			printf("Error at socket(): %d\n", WSAGetLastError());
			freeaddrinfo(result);
			WSACleanup();
			return false;
		}

		// Connect to server
		iResult = connect(ConnectSocket, ptr->ai_addr, (int)ptr->ai_addrlen);

		if (iResult == SOCKET_ERROR)
		{
			closesocket(ConnectSocket);
			ConnectSocket = INVALID_SOCKET;
		}

		freeaddrinfo(result);

		if (ConnectSocket == INVALID_SOCKET)
		{
			printf("Unable to connect to server!\n");
			WSACleanup();
			return false;
		}

		return true;
	};

	// Free the resouces
	void Stop() {
		int iResult = shutdown(ConnectSocket, SD_SEND);

		if (iResult == SOCKET_ERROR)
		{
			printf("shutdown failed: %d\n", WSAGetLastError());
		}

		closesocket(ConnectSocket);
		WSACleanup();
	};
```

```
	// Send message to server
	bool Send(char* szMsg)
	{

		int iResult = send(ConnectSocket, szMsg, strlen(szMsg), 0);

		if (iResult == SOCKET_ERROR)
		{
			printf("send failed: %d\n", WSAGetLastError());
			Stop();
			return false;
		}

		return true;
	};



private:
	char* szServerName;
	SOCKET ConnectSocket;
};

// KOD
void send(void *P)
{
	string msg, msg1, msg2, msg3;
	int i = 0;
	Client1 Client1("192.168.0.100");
	string recived;



	if (!Client1.Start())
		return;

	//msg = "set_digital_out(2, True)\n";

	//a=0.5, v=0.01)\n";
	msg1 = "movel([-5.618518583797006, -2.2833119742908896, 2.201398751276529, -
1.3596290132406592, -0.5043413522775566, -4.652095110072902], a=0.08, v=0.009)\n";
	//a=0.5, v=0.01)\n";
	msg2 = "movel([-5.618518583751086, -2.286711195229915, 2.320497045848828, -
1.4753280869563572, -0.5043413522716449, -4.6520951099786725], a=0.08, v=0.0006)\n";
	//a=1.2, v=0.0015)\n";


	//int imp = 0;
	//imp = kx;
	while (true)
	{
		PressEnterToContinue();
		Client1.Send((char*)msg1.c_str());


		PressEnterToContinue();
		Client1.Send((char*)msg2.c_str());
		go=1;

		PressEnterToContinue();
	}
```

```
	Client1.Stop();

	getchar();
	return;
}
```

regulacija bušenja

```
#define WIN32_LEAN_AND_MEAN

#include <WinSock2.h>
#include <WS2tcpip.h>
#include <iostream>
#include <string>
#include <limits>
#include <algorithm>
#include <bitset>
#include <fstream>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <sstream>
#include <time.h>

// link with Ws2_32.lib
#pragma comment(lib, "Ws2_32.lib")

#define DEFAULT_PORT "30001"
#define DEFAULT_BUFFER_LENGTH     812

//extern double force;
using namespace std;


// Global variable declaration:

extern int rdy;
extern double sensor;
double Fn=5;
extern int go;
extern int go1;

class Client7 {
public:
	Client7(char* servername)
	{
		szServerName = "192.168.0.10";
		ConnectSocket = INVALID_SOCKET;
	}

	bool Start() {
		WSADATA wsaData;

		// Initialize Winsock
		int iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);
		if (iResult != 0)
		{
			printf("WSAStartup failed: %d\n", iResult);
			return false;
		}
```

```
		struct addrinfo     *result = NULL,
			*ptr = NULL,
			hints;

		ZeroMemory(&hints, sizeof(hints));
		hints.ai_family = AF_UNSPEC;
		hints.ai_socktype = SOCK_STREAM;
		hints.ai_protocol = IPPROTO_TCP;

		// Resolve the server address and port
		iResult = getaddrinfo(szServerName, DEFAULT_PORT, &hints, &result);
		if (iResult != 0)
		{
			printf("getaddrinfo failed: %d\n", iResult);
			WSACleanup();
			return false;
		}

		ptr = result;

		// Create a SOCKET for connecting to server
							ConnectSocket =
					socket(ptr->ai_family, ptr-
					>ai_socktype, ptr->ai_protocol);

		if (ConnectSocket == INVALID_SOCKET)
		{
			printf("Error at socket(): %d\n", WSAGetLastError());
			freeaddrinfo(result);
			WSACleanup();
			return false;
		}

		// Connect to server
		iResult = connect(ConnectSocket, ptr->ai_addr, (int)ptr->ai_addrlen);

		if (iResult == SOCKET_ERROR)
		{
			closesocket(ConnectSocket);
			ConnectSocket = INVALID_SOCKET;
		}

		freeaddrinfo(result);

		if (ConnectSocket == INVALID_SOCKET)
		{
			printf("Unable to connect to server!\n");
			WSACleanup();
			return false;
		}

		return true;
	};

	// Free the resouces
	void Stop() {
		int iResult = shutdown(ConnectSocket, SD_SEND);

		if (iResult == SOCKET_ERROR)
		{
			printf("shutdown failed: %d\n", WSAGetLastError());
		}
```

```
		closesocket(ConnectSocket);
		WSACleanup();
	};


	// Send message to server
	bool Send(char* szMsg)
	{

		int iResult = send(ConnectSocket, szMsg, strlen(szMsg), 0);

		if (iResult == SOCKET_ERROR)
		{
			printf("send failed: %d\n", WSAGetLastError());
			Stop();
			return false;
		}

		return true;
	};



private:
	char* szServerName;
	SOCKET ConnectSocket;
};

// KOD
void stop(void *P)
{
	string msg, msg1, msg2;
	int i = 0;
	Client7 Client7("192.168.0.100");
	string recived;



	if (!Client7.Start())
		return;


	msg1 = "movel([-5.618518583797006, -2.2833119742908896, 2.201398751276529, -
	1.3596290132406592, -0.5043413522775566, -4.652095110072902], a=0.08,
	v=0.009)\n";
	msg2 = "movel([-5.618518583751086, -2.286711195229915, 2.320497045848828, -
	1.4753280869563572, -0.5043413522716449, -4.6520951099786725], a=0.08,
	v=0.0009)\n";
	int kk=0;
	int ada=0;
	//double ei=0;

	double v=0;
	double Kp=12;
	double Ki=1;
	double Kd=0.0001;
	double ep=0;
	double ep_old=0;
	double ei=0;
	double ed=0;
	double dt=0.01;
```

```
	while (true)
	{
		while(go==1 && rdy==1 && go1==1)
		{

			ep=Fn-sensor;
			v=Kp*ep+Ki*ei+Kd*ed;
			if(v<0)
				v=0.00000001;
			//cout << "aaaaaa" << endl;
			ofstream myfile;
			myfile.open ("speed.txt", ios::app);
			myfile << v;
			myfile << ",";
			myfile.close();

			string msg11;
			ostringstream msg1;
			msg1.str("");
			//string msg3;
					msg1 << "movel([-5.618518583751086, -
		2.286711195229915, 2.320497045848828, -1.4753280869563572, -
		0.5043413522716449, -4.6520951099786725], a=0.08, v=" << v << ")\n";
			msg11 = msg1.str();
			Client7.Send((char*)msg11.c_str());

			ep_old=ep;
			ep=Fn-sensor;
			ei=ei+ep*dt;
			ed=(ep-ep_old)/dt;

		}
	}

	Client7.Stop();

	getchar();
	return;
}
```

detekcija proboja

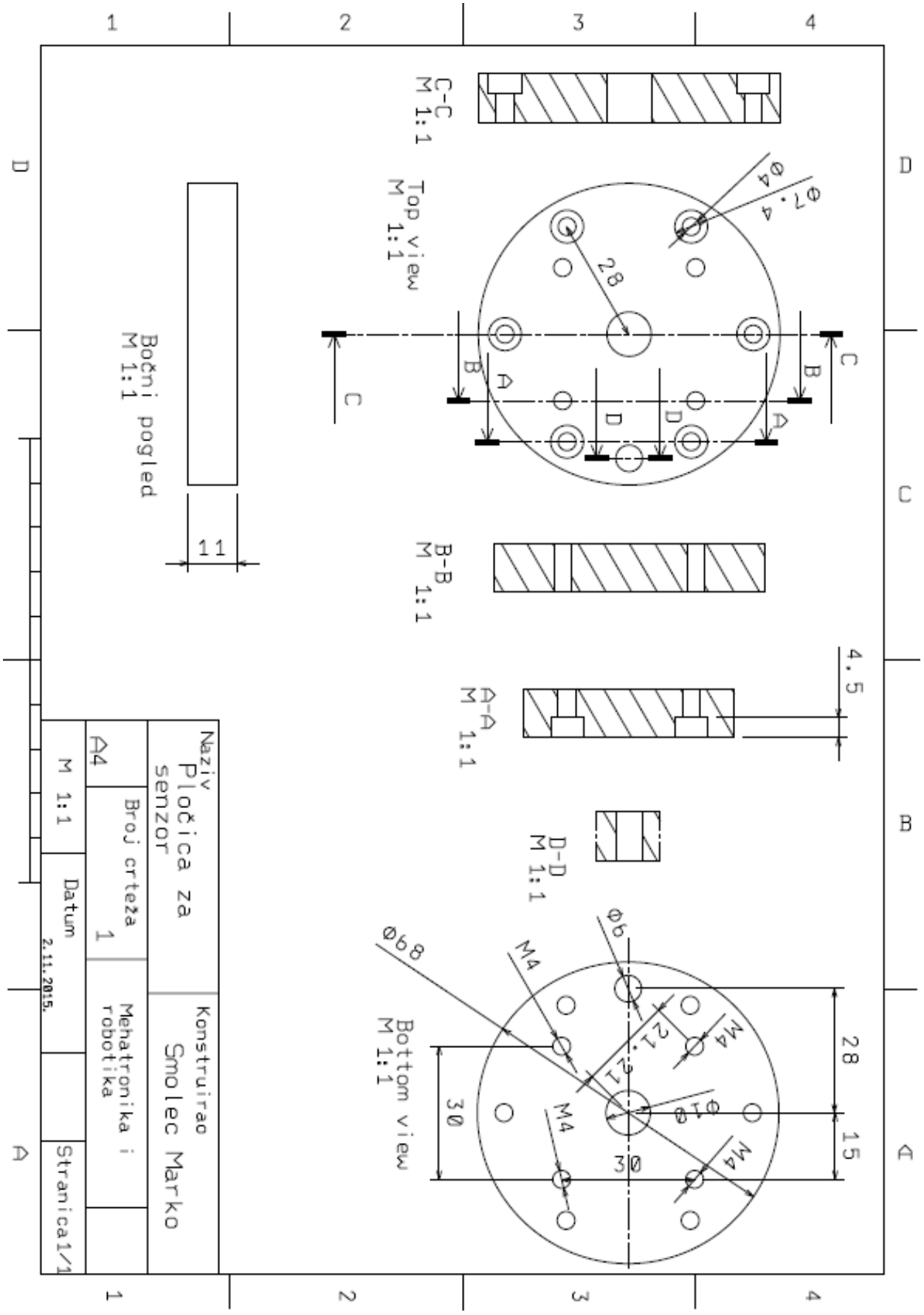
```
#define WIN32_LEAN_AND_MEAN

#include <WinSock2.h>
#include <WS2tcpip.h>
#include <iostream>
#include <string>
#include <limits>
#include <algorithm>
#include <bitset>
#include <fstream>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <sstream>
#include <time.h>

// link with Ws2_32.lib
#pragma comment(lib, "Ws2_32.lib")

#define DEFAULT_PORT "30002"
#define DEFAULT_BUFFER_LENGTH     812
```

```
//extern double force;
using namespace std;


// Global variable declaration:
extern double sensor;
extern int go;
extern int rdy;
int go1=1;

class Client9 {
public:
	Client9(char* servername)
	{
		szServerName = "192.168.0.10";
		ConnectSocket = INVALID_SOCKET;
	}

	bool Start() {
		WSADATA wsaData;

		// Initialize Winsock
		int iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);
		if (iResult != 0)
		{
			printf("WSAStartup failed: %d\n", iResult);
			return false;
		}

		struct addrinfo	*result = NULL,
			*ptr = NULL,
			hints;

		ZeroMemory(&hints, sizeof(hints));
		hints.ai_family = AF_UNSPEC;
		hints.ai_socktype = SOCK_STREAM;
		hints.ai_protocol = IPPROTO_TCP;

		// Resolve the server address and port
		iResult = getaddrinfo(szServerName, DEFAULT_PORT, &hints, &result);
		if (iResult != 0)
		{
			printf("getaddrinfo failed: %d\n", iResult);
			WSACleanup();
			return false;
		}

		ptr = result;

		// Create a SOCKET for connecting to server
				ConnectSocket = socket(ptr->ai_family, ptr-
			>ai_socktype, ptr->ai_protocol);

		if (ConnectSocket == INVALID_SOCKET)
		{
			printf("Error at socket(): %d\n", WSAGetLastError());
			freeaddrinfo(result);
			WSACleanup();
			return false;
		}
```

```
		// Connect to server
		iResult = connect(ConnectSocket, ptr->ai_addr, (int)ptr->ai_addrlen);

		if (iResult == SOCKET_ERROR)
		{
			closesocket(ConnectSocket);
			ConnectSocket = INVALID_SOCKET;
		}

		freeaddrinfo(result);

		if (ConnectSocket == INVALID_SOCKET)
		{
			printf("Unable to connect to server!\n");
			WSACleanup();
			return false;
		}

		return true;
	};

	// Free the resouces
	void Stop() {
		int iResult = shutdown(ConnectSocket, SD_SEND);

		if (iResult == SOCKET_ERROR)
		{
			printf("shutdown failed: %d\n", WSAGetLastError());
		}

		closesocket(ConnectSocket);
		WSACleanup();
	};


	// Send message to server
	bool Send(char* szMsg)
	{

		int iResult = send(ConnectSocket, szMsg, strlen(szMsg), 0);

		if (iResult == SOCKET_ERROR)
		{
			printf("send failed: %d\n", WSAGetLastError());
			Stop();
			return false;
		}

		return true;
	};



private:
	char* szServerName;
	SOCKET ConnectSocket;
};

// KOD
void brzina(void *P)
{
	int i = 0;
```

```
Client9 Client9("192.168.0.100");
string recived;
int set=8;
int count=0;
int check1=0;
int check2=0;
int check3=0;
int check4=0;
       string msg7 = "movel([-5.618518583797006, -2.2833119742908896,
2.201398751276529, -1.3596290132406592, -0.5043413522775566, -
4.652095110072902], a=0.08, v=0.009)\n";
double maxim=8;
int threshold=30;
clock_t t;
double dt=0;
double dt1=0;
double x=0;
int broj_peak=0;
if (!Client9.Start())
       return;


while (true)
{

       while(go==1 && rdy==1)
       {
        Sleep(10);
        if(sensor>maxim)
               {check1=1;
                t=clock();
                dt= (float(t))/CLOCKS_PER_SEC;
                check4=1;
                count=0;}

        while (check4==1 && x<7)
                  {t=clock();
                  dt1= (float(t))/CLOCKS_PER_SEC;
                  x=dt1-dt;
                     cout << x << endl;
                      }

        if(check3==0 && check1==1 && x>3)
        {
               check3=1;

        }

        if (check3==1 && sensor<=maxim)
                              {count++;
                         cout << count << endl;
                             }

        if(check3==1 && check2==0 && count>threshold)
                            {broj_peak++;
                             count=0;}


        if(broj_peak>1)
                        {
                              go1=0;
                              Sleep(100);
```

```
                                Client9.Send((char*)msg7.c_str());
                                check2=1;
                            }

                }
        }

        Client9.Stop();

        getchar();
        return;
}
```


C-C
M 1:1
Top view
M 1:1
Bočni pogled
M 1:1
B-B
M 1:1
A-A
M 1:1
D-D
M 1:1
Bottom view
M 1:1
28
11
4.5
30
15
M4
Naziv
Pločica za senzor
Konstruirao
Smolec Marko
A4
Broj crteža
1
M 1:1
Datum
2.11.2015.
Mehatronika i robotika
Stranica 1/1